NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

FEATURE-BASED LOCALIZATION IN SONAR-EQUIPPED AUTONOMOUS MOBILE ROBOTS THROUGH HOUGH TRANSFORM AND UNSUPERVISED LEARNING NETWORK

by

Jonathan Scott Glennon

June, 1998

Thesis Advisor: Xiaoping Yun

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1998.	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE FEATURE-BASED LOCALIZATION IN SONAR-EQUIPPED AUTONOMOUS MOBILE ROBOTS THROUGH HOUGH TRANSFORM AND UNSUPERVISED LEARNING NETWORK

5. FUNDING NUMBERS

6. AUTHOR(S) Jonathan Scott Glennon

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

As we approach the new millennium, robots are playing an increasingly important role in our everyday lives. Robotics has evolved in industrial and military applications, and unmanned space exploration promises the continued development of ever-more-complex robots. Over the past few decades, research has focused on the development of autonomous mobile robots – robots that can move about without human supervision. This brings with it several problems, however, specifically the problem of localization. How can the robot determine its own position and orientation relative to the environment around it?

Various methods of localization in mobile robots have been explored. Most of these methods, however, assume some a priori knowledge of the environment, or that the robot will have access to navigation beacons or Global Positioning Satellites. In this thesis, the foundations for feature-based localization are explored. An algorithm involving the Hough transform of range data and a neural network is developed, which enables the robot to find an unspecified number of wall-like features in its vicinity and determine the range and orientation of these walls relative to itself. Computation times are shown to be quite reasonable, and the algorithm is applied in both simulated and real-world indoor environments.

14. SUBJECT TERMS Autonomous mobile robots, Hough transform, localization, Nomad Scout mobile robot, competitive neural networks, data clustering.

15. NUMBER OF PAGES 109

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

FEATURE-BASED LOCALIZATION IN SONAR-EQUIPPED AUTONOMOUS MOBILE ROBOTS THROUGH HOUGH TRANSFORM AND UNSUPERVISED LEARNING NETWORK

Jonathan Scott Glennon
Captain, United States Marine Corps
B. S., United States Naval Academy, 1990

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June, 1998

ABSTRACT

As we approach the new millennium, robots are playing an increasingly important role in our everyday lives. Robotics has evolved in industrial and military applications, and unmanned space exploration promises the continued development of ever-more-complex robots. Over the past few decades, research has focused on the development of autonomous mobile robots – robots that can move about without human supervision. This brings with it several problems, however, specifically the problem of localization. How can the robot determine its own position and orientation relative to the environment around it?

Various methods of localization in mobile robots have been explored. Most of these methods, however, assume some a priori knowledge of the environment, or that the robot will have access to navigation beacons or Global Positioning Satellites. In this thesis, the foundations for feature-based localization are explored. An algorithm involving the Hough transform of range data and a neural network is developed, which enables the robot to find an unspecified number of wall-like features in its vicinity and determine the range and orientation of these walls relative to itself. Computation times are shown to be quite reasonable, and the algorithm is applied in both simulated and real-world indoor environments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This research was only possible due to the efforts and sacrifices of many people. An enormous debt of gratitude is owed to my advisor, Dr. Xiaoping Yun, for his advice, guidance, leadership, and inspiration. Also to my second reader, Dr. R. Gary Hutchins, for his wisdom and counsel. Both have made the conduct of the research and the writing of this document a thoroughly rewarding and enjoyable learning experience.

Many thanks to the faculty and staff of the Electrical and Computer Engineering Department at the Naval Postgraduate School who, through their patience and example, have given me the benefit of their wisdom and experience. In particular, Dr. Monique Fargues was a very helpful source of information regarding neural networks. Thanks also to the staff of the Dudley Knox Library.

Thanks go out to the United States Marine Corps, who saw fit to send me to the Naval Postgraduate School, and to the taxpayers who bore the financial burden for my education. Appreciation goes to the wonderfully creative people at Nomadic Technologies in Mountain View, California; and also to the makers of MATLAB.

Finally, and most importantly, I would like to thank my wife, Monika, and our children, Jackie and R. J. They have endured the countless hours I have had to spend away from them without a single complaint. This effort would not have been possible without their infinite love, support, and patience.

I. INTRODUCTION

Twelve years ago, the number four reactor at Chernobyl became unstable, and the worst nuclear accident in history became a reality. The concrete encasement built around the facility is beginning to show signs of structural failure, yet the site is still so radioactive that people cannot work inside. But "Pioneer," a thousand-pound robot designed by RedZone, was designed to enter the hazardous area, and take readings and images to help scientists assess the load bearing performance of the walls. [Refs. 1, 2]

This scenario demonstrates one of the reasons that robots have begun to appear in almost every facet of our lives, often emerging as a popular alternative to human labor. Robots are able to enter hazardous areas and perform tasks in environments where humans would be placed in unacceptable danger. If the robot becomes a casualty, only a financial loss is suffered. This makes them ideally suited to enter areas like Chernobyl, and other environments as well. For 80 days, "Sojourner" was able to send information to NASA about the surface of the planet Mars. It would have been difficult to sustain a human exploration team for this length of time in a radiation environment where temperatures range from -13.3 to 54.4 degrees Celsius [Ref. 3]. The risks of such a mission would have been extraordinary. NASA was able to use robots on this primary exploration, thereby reducing risk for a planned human exploration in the future.

Clearly, the inherent suitability of robots in hazardous environments offers many potential military applications for mobile robots. Some potential military applications have recently been explored at the Naval Postgraduate School, including a navigation system for an outdoor robot [Ref. 4], and battlefield surveillance [Ref. 5]. Applications involving the cooperation of multiple robots are also being explored [Refs. 5, 6]. A rugged platform has been used in development for possible applications in searching for mines and unexploded ordnance [Ref. 7]. The Defense Advanced Research Projects Agency (DARPA) supports research in distributed robotics for military applications [Ref. 8].

In addition to reducing risk to human life, robots offer other advantages. The human mind has a limited attention span. Humans are prone to boredom and fatigue. Robots are immune to boredom and fatigue (as long as power supplies are maintained).

Thus, robots are better suited than humans to perform repetitive or tedious tasks. Searching an area is a prime example. The accuracy of the search inherently depends on the searcher's level of concentration, but concentration will degrade as a human searcher becomes fatigued and bored. A robot can perform the same task with uniform accuracy, and can continue the search for days or weeks without taking a break. Robots are also ideally suited for tasks such as sentry duty or security patrols, since these tasks involve alertness during a repetitive behavior.

Given the proper array of sensors, the potential applications for robots is boundless. But clearly the need is greatest for intelligent, mobile machines; especially those which can work in concert and adapt to dynamic environments. But sensors alone do not enable a robot to react to its environment. The information from these sensors must be processed in clever ways. Some degree of on-board intelligence is crucial to this processing.

Some robots are hard-mounted, while others have mobility. For those which have mobility, it becomes a challenge to enable the robot to move about unsupervised and unassisted. Such robots are often called *autonomous mobile robots.* Controlling the robot's movement with a joystick or other interface is common practice today, but this does not represent the ideal solution. It would be better if the robot could autonomously navigate itself. But this brings several issues with it, not the least of which is localization.

A. THE LOCALIZATION ISSUE

One very important issue associated with the operation of an autonomous mobile robot is localization. Specifically, localization is the process of defining the real-time position and orientation of the robot with respect to a given coordinate system. Usually, this coordinate system is some representation of the real world. Localization is important; the robot must know where it is in order to navigate to a new location. The robot must have a point of reference if it is to explore an area.

When operating in outdoor environments (on earth),the availability of Global Positioning Satellites (GPS) makes the localization problem a relatively simple one to

solve, but by no means is GPS a universal solution. A robot that operates in an indoor environment is often unable to receive GPS signals. Even certain outdoor environments near buildings or dense vegetation may leave the robot without simultaneous connections to four satellites. If the robot is to complete its task outside of Earth's atmosphere (the surface of Mars, for example), GPS offers no assistance whatsoever. Even in applications where GPS is available, the accuracy of the commercially available signal may not be sufficient for some applications.

Dead reckoning is probably the simplest and most common method of self-localization in mobile robots. Generally, some type of *odometer* is used to count wheel rotations or measure distance traveled or velocity. Given the initial placement, and velocity as a function of time, the robot computes its present location by integrating its velocity function. The nature of integration, however, implies that imperfections in the odometry readings will accumulate over time. Thus, the longer a robot moves about in a particular environment, the less accurate its localization will be. This is the prime disadvantage of dead reckoning. In the case of wheeled robots, wheel slippage is a common cause of dead reckoning errors. Dead reckoning is not a viable means of localization for long term mobility, unless some means exists to check for and correct cumulative errors.

Beacon tracking is another way to enable localization, and early work at the Naval Postgraduate School focused on beacon tracking robots [Ref. 9, page 14]. A number of beacons are placed somewhere in the robot's vicinity at known locations. The beacons transmit signals, which are received and then interpreted by the robot. Either through triangularization or some other method, the robot determines its position relative to these known locations. This method becomes unfeasible if the environment cannot be equipped with beacons. The surface of Mars, or a battlefield, for example, cannot be equipped with navigation beacons prior to the deployment of the robot.

To enable an autonomous mobile robot to operate in a given environment over an extended period of time, without assuming the availability of GPS satellites or navigation beacons, it is clearly necessary to correct the dead-reckoning errors that accumulate over time. One general approach to correcting dead-reckoning errors is to provide the robot

with the ability to recognize physical features or landmarks in its environment as it moves about, and to calibrate itself with respect to these features when it arrives at the previously traveled area again. This method is commonly referred to as *feature-based* localization, and it is very similar to the method that humans use.

This thesis explores the foundations for such a method. I have chosen to limit the investigation to the very specific (and simple) case of an autonomous mobile robot equipped with time-of-flight sonar range sensors, restricted to an indoor environment. The features that the robot learns to recognize will be walls. It is my hope that if the foundations can be established in this simplified case, they can later be modified and applied to more complex scenarios.

B. GOALS OF THE THESIS

In this thesis, an algorithm is to be developed which enables a robot to recognize an undetermined number of wall-like features in its environment. The algorithm will determine the number of walls, as well as the range to and orientation of these walls relative to the robot.

The objective is to develop an algorithm which allows the robot to automatically determine the range to and orientation of any suitable walls, without a priori knowledge of the environment. The algorithm must be tested and shown to perform reasonably well in real indoor environments.

It should be noted that this thesis is intended to demonstrate the concept of feature-based localization. To facilitate the concept demonstration, data are processed off-line. In practice, the same algorithm should be run in the high-level control of the robot itself. Running the algorithm on the robot would prevent close analysis of the performance of the algorithm, however, so this step is left for future research.

C. THESIS OUTLINE

The remainder of this thesis deals with the introduction and investigation of an algorithm for finding an undetermined number of walls in an autonomous mobile robot's environment. Some background information on the robotic system used for testing is clearly necessary, as well as some information on the Hough transform and competitive neural networks.

Chapter II gives a system overview of the mobile robot used in this experiment; the NOMAD SCOUT. This robot is a product of Nomadic Technologies, Inc.

In Chapter III the problem is presented in more detail. Related projects are discussed in the literature survey. A solution involving the Hough transform and a competitive neural network is proposed.

Since the reader may not be familiar with the Hough transform, an introduction is provided in Chapter IV. The grid-based approach to extracting data from the Hough domain is discussed. The key elements of the Hough domain are pointed out, and solved for explicitly.

An introduction to competitive neural networks is provided in Chapter V. The Kohonen neural network is introduced, and several modifications are made to it. It is shown how such a network may be applied to find an unspecified number of clusters in a given pattern space. A demonstration of clustering in a two-dimensional pattern space is provided, and the time required to conduct the clustering is shown to be adequate.

The specific implementation of the proposed approach is discussed in Chapter VI. The implementation is discussed in several stages. The Hough transformation is implemented in an unconventional manner, yielding clusters of points in the Hough domain which represent groups of curve intersections. These clusters are then classified by the network of Chapter V, and each cluster is represented by an exemplar vector. These exemplar vectors are taken to be representations of the walls near the robot.

The results of the algorithm applied to both simulated and real-world sonar returns are given in Chapter VII. Several scenarios are discussed in detail. The results show the algorithm to perform adequately for the chosen application.

In Chapter VIII, the implications and potential applications of the proposed algorithm are discussed. Additionally, potentials for future work are identified and discussed.

The code used to implement the proposed algorithm is included in the appendices to this document. Appendices A, B, and C are the implementation of the algorithm itself, while appendix D is the code used to gather the sonar data.

II. SYSTEM OVERVIEW

Several groups are pursuing research relating to autonomous mobile robots at the Naval Postgraduate School. The school has purchased a number of robots from a nearby supplier to facilitate this research. Nomadic Technologies, Inc., located in Mountain View, CA, is the producer of several models of autonomous mobile robots.

The school has purchased one NOMAD 200 ™ mobile robot, and four NOMAD SCOUT ™ robots, with several more Scout robots planned for the near future. These platforms are, for the most part, code-compatible [Ref. 10]. The algorithm outlined in this thesis was tested in real conditions on the Scout platform. This Chapter is intended to give the reader some familiarity with the platform, and provide a technical context for comparisons.

Only those aspects of the platform which pertain to the thesis are explained in this Chapter. A complete description of the Scout platform can be found in References [10 and 11].

Off-line processing of the range data gathered by the robot was conducted on a computer, using MATLAB ™ version 4.2 (b), a software package by Mathworks, Inc. A brief description of the computer platform and the software package is included to provide further technical context for comparisons.

A. SYSTEM OVERVIEW: NOMAD SCOUT ™

The Scout is an autonomous mobile robot system, equipped with a variety of sensors. 16 independent ultrasonic time-of-flight sonar sensors are included in the package for range-finding, effective over 6 to 255 inches. 16 independent tactile switches are located about the circumference of the robot. An odometry sensor is included to facilitate dead reckoning. The control system is hierarchical. The majority of the "low-level" or "housekeeping" control functions, including the sensing and communications, are performed by an on-board Motorola MC68332 multiprocessor. Motor control is conducted by a TMS320C14 DSP chip. "High-level" controls can be administered either by a laptop computer mounted on top of the robot, or a remote Linux or Unix workstation

connected to the robot via a radio modem. The Scout is powered by two 12 Volt, 17 Ampere Hour lead-acid batteries, which can power the robot for up to 20 hours of normal operation when fully charged. [Refs. 10, 11].

Most applications conducted at the Naval Postgraduate School are administered from a Unix workstation via a wireless modem. In this particular application, a Unix workstation was connected to the robot via wireless modem only to gather range data from the sonar sensors. The data was then saved in ASCII format, and subsequently processed off-line. Hardware and software platforms used in this off-line processing are described later in this chapter.

1. Mechanical Description

The newest version of the Scout is shown in Figure 1. Without its batteries, the Scout weighs 23 kilograms. It is 34 centimeters tall, and 38 centimeters in diameter. [Ref. 11]

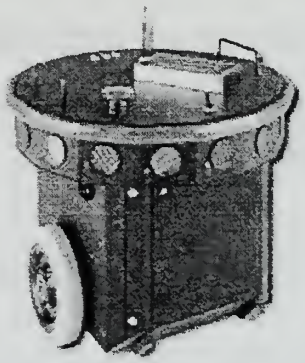

Figure 1. Nomad Scout II (from Ref. [12])

The platform can travel with a maximum velocity of 1.0 meter per second, and can accelerate at up to 2 meters per second squared. The ground clearance is 3.5 centimeters. The Scout is a 2 degree-of-freedom robot with 2 wheel differential drive at the geometric center of the robot. [Refs. 10, 11]

2. Odometry

The Scout is able to keep a running, real-time integral of its current position in world coordinates. It assumes its startup position to be the origin, unless the origin is reset during operation. The *x* axis extends from the center of the robot to the forward direction (the direction the robot is facing). The *y* axis extends from the center of the robot to the robot's left. The robot also tracks its orientation, given relative to the *x* axis, such that counter-clockwise is a positive angle.

The odometric encoder has finite resolution. The translational movements (relative to the *x* and *y* axes) are measured by 167 counts per centimeter, and returned at one tenth of an inch resolution. Orientation is measured with 45 counts per degree, and returned with one tenth of a degree resolution. [Refs. 10, 11]

3. The Sensus 200 ™ Sonar Ranging System

Sensus 200™ is the trademark name given to the time-of-flight sonar ranging system installed on the Scout by its creators at Nomadic Technologies. It consists of 16 independent channels equally dispersed about the circumference of the robot. The separation of the center axes of adjacent sonar channels is 22.5 degrees. The sensors used are standard Polaroid transducers, driven by a Polaroid 6500 ranging board. Each transducer has an independent beam width of 25 degrees, so there is some overlap. [Ref. 13]

The system is a time-of-flight ranging sensor, based on the return time of an ultrasonic acoustic signal. At the initiation of each read cycle, each transducer sequentially transmits a pulse at 49.4 kHz, and the time required for an echo to be received is measured. The transducers have a tendency to ring after transmitting, so the echo receivers must be blanked for a certain amount of time. This results in a minimum distance of about 6 inches. If no echo return is detected, the next read cycle is initiated, resulting in a maximum distance of 255 inches. [Ref. 13]

Under ideal operating conditions, the sensor array would be expected to return accurate range findings over 6 to 255 inches, and do so with 1 percent accuracy over the

entire range [Ref. 10]. In practice, however, the performance of the time-of-flight sonar operating alone is less reliable. This is due primarily to the non-ideal propagation characteristics of acoustic signals. Echo returns tend to be accurate only when they are reflected by a nearly orthogonal surface. Signals may be reflected by more than one surface before returning to the echo receiver, resulting in a range finding that is higher than the true value. The material construction of the reflective surface may also tend to damp acoustic signals, giving (in the worst case) a maximum range finding when in fact the surface is much closer. There are a number of ways to improve the reliability of the range data, and some of these will be discussed in Chapter VIII.

B. COMPUTER AND SOFTWARE

As stated earlier, the range data were simply collected by the robot, and processed off-line in order to demonstrate the concept of feature-based localization in a meaningful way. The data were ported in ASCII format to a Gateway 2000 ™ P5-75 system. The processor is a first generation Intel™ Pentium ™, with 75 MHz clock speed. 32 megabytes of random access memory are installed in the system. The operating system used is Microsoft™ Windows 95 ™.

Data were processed using original programs written for the Mathworks software program MATLAB™, version 4.2 (b). No other programs were running when data analysis was conducted.

C. CHAPTER SUMMARY

In this chapter, a brief overview of the Nomad Scout robotic platform was provided. A brief description of the platform used to process data off-line was also given. The next chapter will introduce the problem of localization in further detail, and outline the proposed approach.

III. PROBLEM STATEMENT AND PROPOSED APPROACH

This chapter begins by defining the problem to be solved. Next, related literature is surveyed and summarized. Finally, the proposed solution to the problem is outlined, and broken into steps. These steps are then further detailed in the following chapters.

A. PROBLEM DEFINITION

As stated in Chapter I, this thesis investigates the problem of feature-based localization of an autonomous mobile robot. Localization is the process of ascertaining the real time position and orientation of the robot relative to a world coordinate system. Localization becomes an important issue, because a robot cannot possibly navigate or explore in an environment without accurate localization.

Many researchers have studied the problem of localization, and literature is widely available. Some of these methods assume the a priori availability of a world map. Others use beacons in the environment that can be recognized by the appropriate sensors installed on the robot, but this approach is not always feasible. Global Positioning Satellites can be used in some situations, but not all.

Dead reckoning is a very common feature, found on many mobile robots. It is insufficient, because odometric errors accumulate over time. Experience with mobile robots at the Naval Postgraduate School shows that substantial errors can accumulate in as little as 30 minutes, as shown in Figure 2. The figure shows the results of a Scout robot used to map an indoor environment. The actual indoor environment is a laboratory at the Naval Postgraduate School; in reality the walls should be nearly orthogonal or parallel to one another. The odd corridor at the top of the map reflects odometric errors which have accumulated in only 31 minutes. Clearly the need arises for some means to identify and correct cumulative odometric errors.

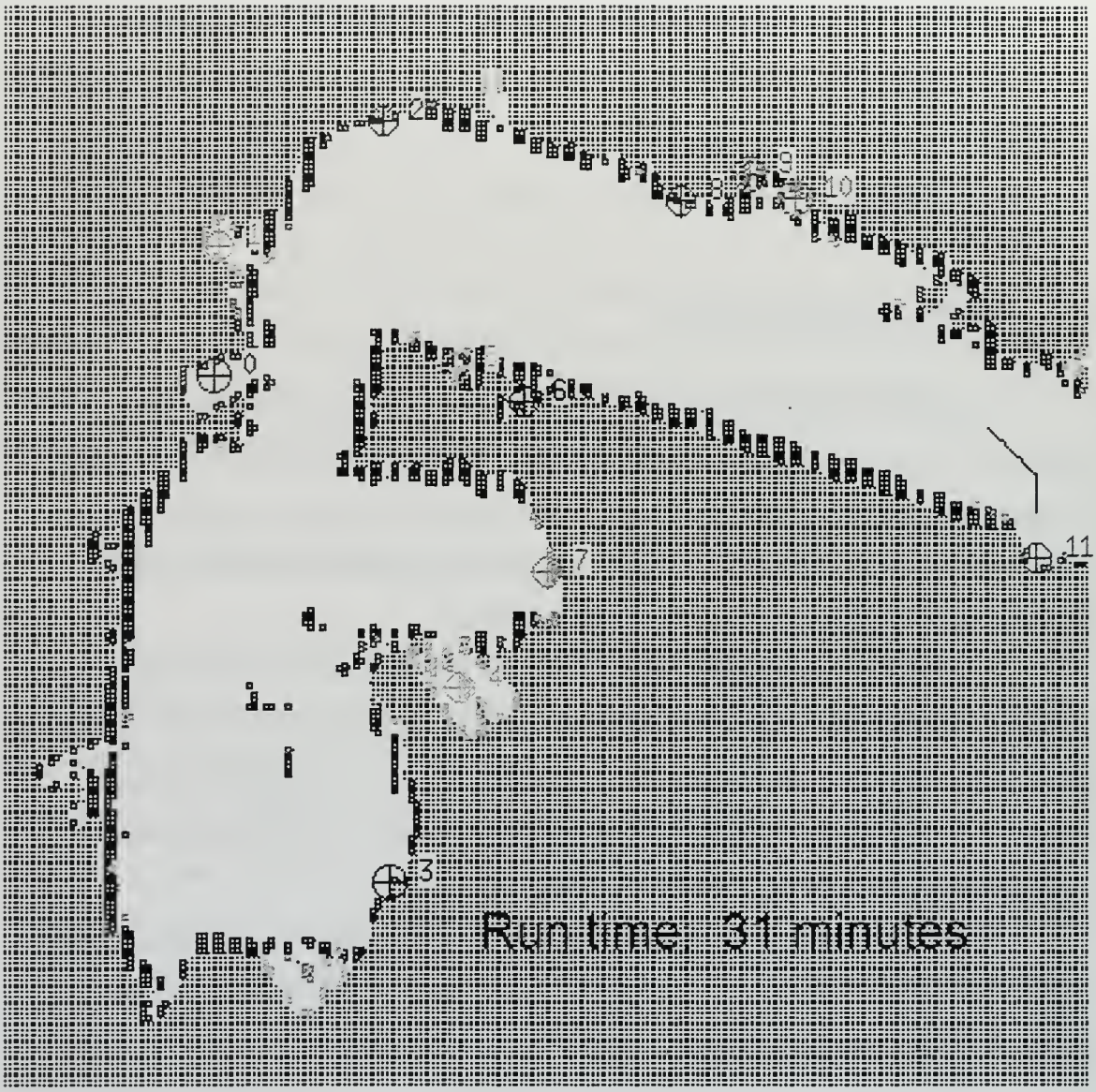

Figure 2. Dead reckoning error

If the robot could identify features in its environment, then it would be possible to correct these cumulative errors. A robot could identify features near it at startup, before any errors have occurred. Then, after it has moved about for a period of time, it would return to this location and look once again for those features. If the features appear in a slightly different location or orientation during this second sample, then the difference must be due to cumulative dead reckoning error. The robot simply calibrates its x and y position and orientation to compensate for the difference.

This is a difficult problem, and this thesis does not attempt to solve it in its entirety. Rather, the problem is simplified with certain assumptions, in the expectation that fundamental concepts explored herein can be extended to more complex scenarios. First, we assume that the robot is indoors. Second, we assume that the features to be recognized are nearly straight lines; in other words, walls. Finally, we assume that the robot is equipped with some type of ranging sensor; in our case we are using an ultrasonic sonar array. The specific platform used to gather range data and explore the concept of feature-based localization is the Nomad Scout mobile robot described in Chapter II.

Armed with these assumptions, the next step is to define an achievable objective. The intent is only to demonstrate the feasibility of the concept of feature-based localization. Hence, the problem becomes one of enabling the robot to search for and find an undetermined number of walls in its vicinity, and uniquely and accurately determine the position and orientation of these walls relative to itself. This process must be conducted without a priori knowledge of the environment, and the results must be based entirely on the range data from the sonar array. If this can be done, we have accomplished the goal of demonstrating a means for correcting cumulative dead reckoning errors. We simply conduct the search twice; once at startup at which time any nearby walls are identified and stored in memory. After the robot has moved about the room and accumulated some dead reckoning error, we send the robot back to the dead reckoning origin, facing in the direction of the dead reckoning x axis. The features (walls) will appear in the second search with a slightly different position and orientation, and the difference reflects the cumulative error due to dead reckoning. Corrections are made to the robot's dead reckoning localization until the features appear in their original position and orientation.

Finally, we note that any straight wall can be uniquely described from any point near the wall using only two parameters, as shown in Figure 3. If this "point" is the center of the robot, then the first parameter is the shortest distance to the wall; in other words, the distance from the center of the robot to the wall along a line orthogonal to the wall. The second parameter is the orientation of the wall; in other words, the counter-clockwise angle from the forward direction of the robot to the orthogonal line.

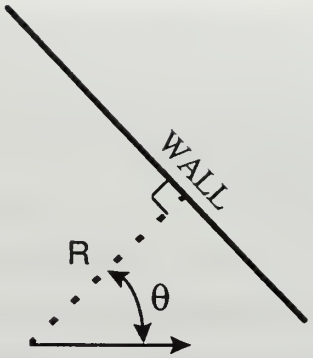

Figure 3. Parameters used to describe a wall

B. LITERATURE SURVEY

In 1962, Hough developed a means for representing complex arrangements in the Cartesian domain by parameters [Ref. 14]. In many cases, the complex arrangements were transformed to much simpler arrangements. The parameters used were the traditional polar axes, radius and counter-clockwise angle from the x axis. This transformation by parameterization became known as the Hough transform.

Throughout the 1960s and 1970s, competitive neural networks were developed by many researchers, including Stephen Grossberg, Cristoph von der Malsburg, and Tuevo Kohonen, among others [Ref. 15]. In particular, Kohonen introduced a simplified version of the *INSTAR* learning rule to be used with competitive networks. This simplified rule is often referred to as the Kohonen learning rule [Refs.15, 16].

In 1972, Duda and Hart suggested a means for using the Hough transform to detect lines in pictures [Ref. 17]. The paper has become a standard reference, not only for researchers studying localization of sonar-equipped mobile robots, but also for those studying robot vision and image processing. The method proposed by Duda and Hart for extracting key information from the Hough domain is outlined in Chapter IV. Alternative methods using neural networks inspired by Kohonen have since been presented [Ref. 18].

A team of researchers in Sweden have recently begun to experiment with the Hough transform as a feature-recognition tool in a mobile robot [Refs. 19, 20, 21]. In this case, the Hough transform is modified slightly; range findings at certain values are

weighted more heavily than others. The "Range Weighted Hough Transform" (RWHT) is applied inside the feedback loop of a mobile robot to assist in localization with documented results.

Variations on the Hough transform have also been used to find shapes other than straight lines. The methodology originally proposed by Duda and Hart has been modified and used to detect curves using a Fourier parameterization [Ref. 22]. By combining the Hough transform with a neural network, a complete shape recognition system has been proposed [Ref. 23].

Previous work at the Naval Postgraduate School explored localization techniques for a Nomad 200 mobile robot using the Hough transform of ultrasonic sonar range findings [Ref. 24]. The proposed algorithm was successful in finding the longest wall and determining the range and orientation of this wall. The method used for analysis of the Hough domain was very similar to that proposed by Duda and Hart.

At the Naval Research Laboratory in Washington, D. C., a group of scientists have been investigating the feasibility of simultaneous localization and exploration. In general, it is necessary to have accurate localization in order to facilitate exploration. At the same time, most methods of localization assume some a priori knowledge of the environment. This apparent contradiction is elegantly addressed through the use of evidence grids. The concept is to divide the environment into small grid squares, and look for evidence that a particular square may or may not be occupied. [Refs. 25, 26]

The Hough transform has proven to be a very versatile tool, and has been put to many other applications in recent years. A three dimensional imaging system using laser generated ultra short x-ray pulses was developed in 1997 [Ref. 27]. A non-invasive iris recognition system employing the Hough transform was developed in 1996 [Ref. 28].

C. PROPOSED APPROACH

The Nomad Scout mobile robot is equipped with an ultrasonic sonar array. This enables the robot to take range findings in all directions, and develop a two-dimensional view of the world in terms of the range returns. We will take the center of the robot at

startup to be the origin of a Cartesian plane, as in Figure 4. The x axis is taken to be the initial forward-looking direction of the robot. The y axis is taken to be the direction 90 degrees counter-clockwise from the robot. This coordinate frame is commonly called the *robot coordinate frame*, as opposed to the *world coordinate frame*, in which the origin and axes are fixed with respect to the world.

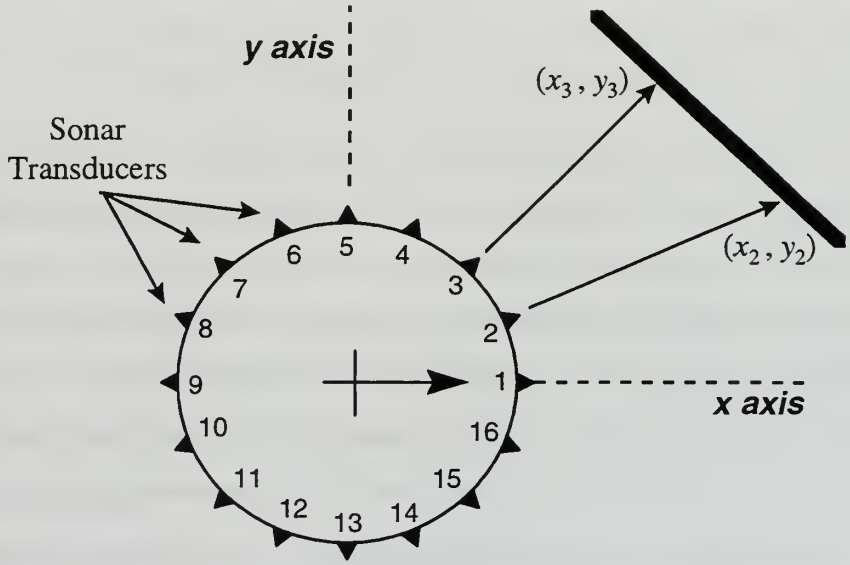

Figure 4. Cartesian representation of world coordinate system

Note that at startup, the world coordinate system and the robot coordinate system are the same. This will not likely be the case when the robot returns to this location a second time for new readings, as some dead reckoning error will have accrued. In each instance, the robot will determine the range and bearing to any nearby walls in terms of *robot coordinates*. If the algorithm is able to provide reliable findings for the ranges and bearings of these walls, then any substantial difference must be due to the dead reckoning errors resulting from wheel slippage and other factors.

Each range finding from one of the sonar transducers can be regarded as a point in the Cartesian plane, uniquely described by an (x, y) pair. The proposed algorithm will be broken into the following four steps:

1) The range data returned from the Sensus 200 system will be converted to (x, y) pairs representing the Cartesian points (in robot coordinates) where the echo occurred.
2) These (x, y) coordinates will be parameterized using Hough parameters.
3) Regions in the Hough domain where curves tend to intersect each other must be represented by clusters of points.
4) A competitive neural network will be employed to identify clusters and represent them by a single point.

D. CHAPTER SUMMARY

In this chapter, the problem of localization in mobile robots was discussed and defined. Some of the past and present research in related topics was discussed. A means for implementing feature-based recognition was proposed and broken into steps. The following two chapters will introduce the tools necessary to complete the steps outlined in this chapter.

IV. THE HOUGH TRANSFORM

The Hough transform was filed as a U.S. patent in 1962 [Ref. 14], and has since been introduced into more standard technical literature by numerous sources. It has also been called the point-to-curve transformation [Refs. 17, 29], since the method entails mapping each point in the Cartesian space into a curve in the *parameter space*. We begin this chapter by introducing the fundamentals of parameterization, and then discuss specifically the Hough parameterization.

A. FUNDAMENTAL PARAMETRIC REPRESENTATION

To demonstrate the concept of parameterization, we begin with a simple example. It is one of the fundamental concepts of algebra that a straight line in the Cartesian plane can be uniquely and completely described by two parameters only; the slope of the line and the *y*-intercept. The parameterization is already familiar to the reader as:

$$y = m_o x + b_o \qquad (1)$$

where m_o is the slope of the line and b_o is the *y*-intercept. In this case, the parameters are m_o and b_o.

If this line is plotted in the *m-b* parameter plane, it is transformed into a single point. Further, it can be seen that for any point (x_o, y_o), the set of all lines through (x_o, y_o) will be transformed into a straight line in the *m-b* parameter plane (see Figure 5 (a)). This is not so surprising since the transformation equation (Equation 2) demonstrates a clearly linear relationship between m and b when x_0 and y_0 are held constant.

$$b = -x_0 m + y_0 \qquad (2)$$

It can also be seen that if several points in the Cartesian plane lie on a line, then these points will be transformed in the *m-b* parameter plane as lines which all intersect at a single (m,b) point (see Figure 5 (b)). Not surprisingly, the (m,b) coordinates of this intersection point are exactly the slope and *y*-intercept of the original line through the points in the *x-y* coordinate plane.

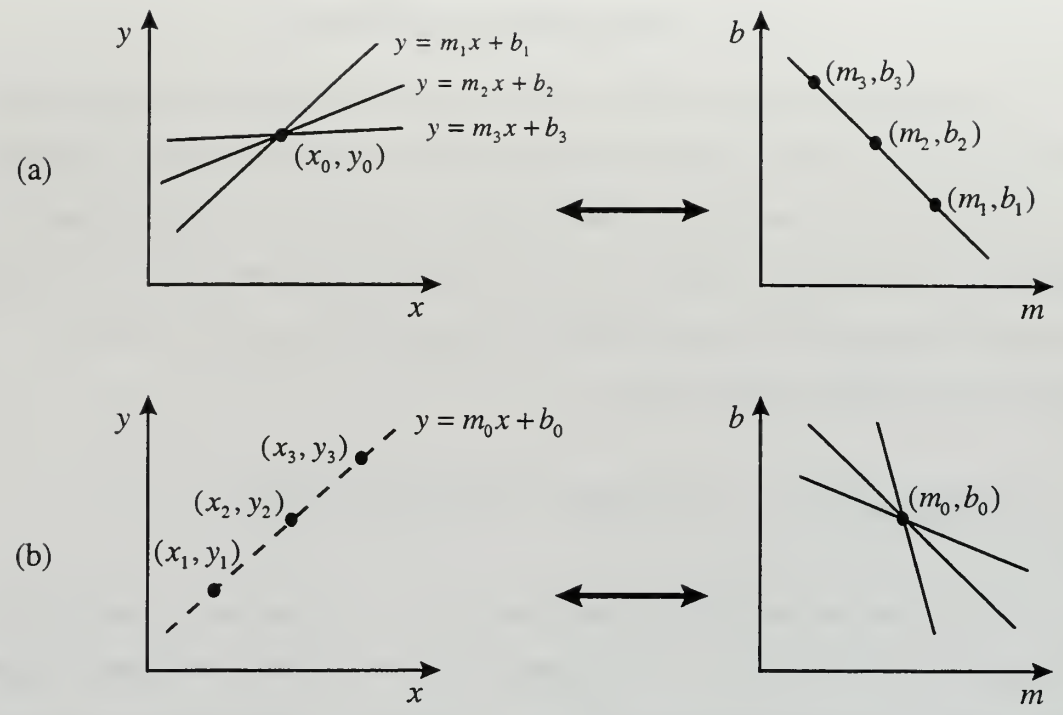

Figure 5. Point - Line transformations (after Reference [24]). (a) A point in the Cartesian plane is transformed to a line in the parameter plane. (b) A line in the Cartesian plane is transformed to a point in the parameter plane.

One drawback in this particular parameterization is that a singularity exists. The independent variable in the *m-b* parameter plane is the slope, which is unbounded. When a line in the Cartesian plane is parallel to the *y*-axis, the slope becomes infinite.

B. POINT-CURVE TRANSFORMATION

As described by Hough [Ref. 14], and later by Duda and Hart [17], another set of parameters can be used to transform the Cartesian plane into the θ-ρ plane. The line described earlier by Equation 1 could also be described by

$$\rho_0 = x\cos\theta_0 + y\sin\theta_0 \qquad (3)$$

where ρ_0 is the shortest distance from the origin to the line, and θ_0 is the angle of the normal to the line through the origin (see Figure 6). In this case as well, a straight line in the Cartesian plane is uniquely and completely described by just two parameters, θ and ρ, but in this new parameterization the singularity problem incurred in the *m-b* parameterization has been eliminated. Since both parameters used to describe the line are

defined by the *normal* to the line through the origin, this parameterization is called the *normal parameterization.*

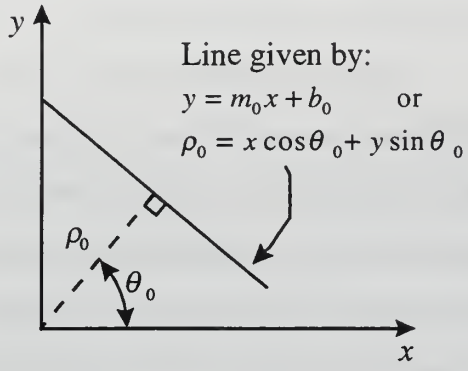

Figure 6. Normal parameterization of a line

The normal parameterization demonstrates some interesting properties when the transformation is plotted, as shown in Figure 7.

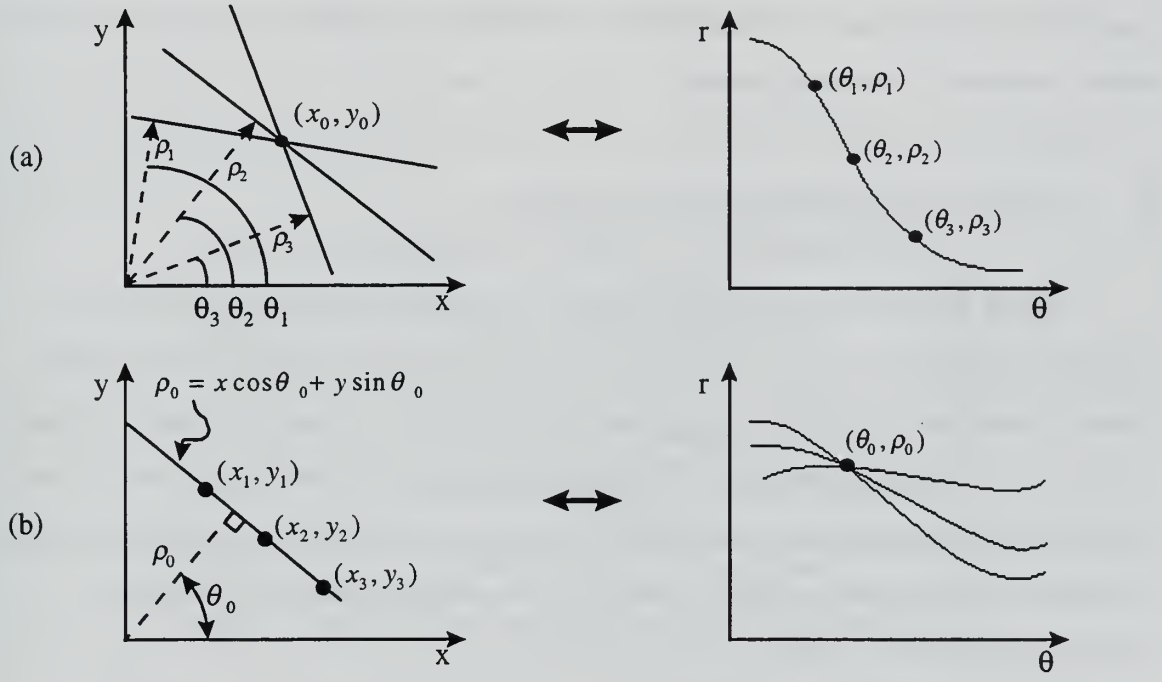

Figure 7. Point-curve transformations (after Reference [24]). (a) A point in the Cartesian plane is transformed to a curve in the normal parameter plane. (b) A line in the Cartesian plane is transformed to a point in the normal parameter plane.

Equation 3 serves as the transformation equation, where the independent variable θ varies over the range (-π,π). For any given point (x_o, y_o) in the Cartesian plane, the set of all lines passing through (x_o, y_o) transforms into a sinusoidal curve in the (θ, ρ)

parameter plane (see Figure 7 (a)). For this reason, the transformation resulting from normal parameterization is sometimes called the *point-curve transformation*. The curve resulting from the transformation of (x_o, y_o) is given by

$$\rho_0 = x_0 \cos\theta + y_0 \sin\theta \qquad (4)$$

From Figure 7 (b) it can also be seen that *collinear* points in the Cartesian plane will each be transformed into curves, and that these curves will have a single point of intersection in the (θ, ρ) parameter plane. Furthermore, the (θ, ρ) coordinates of this intersection point are precisely the normal parameters; θ is the angle of the normal and ρ is the shortest distance from the origin to the line (i. e. the length of the normal).

The Hough transform uses the normal parameterization and point-curve transformation described. The centerpiece the Hough transform and the shape recognition algorithm proposed by Duda and Hart [Ref. 17] is that a line in the Cartesian plane will be transformed to a single point. The task of recognizing a line has now been reduced to the task of finding a point.

C. HOUGH TRANSFORM TECHNIQUES

Given that lines in the Cartesian plane can be reduced to points in the Hough domain, the challenge comes in finding the points where curves intersect in the Hough domain. Particularly challenging is the task of automating this process so that a computer or robot can find these points without human assistance. The reader should bear in mind that if the data in the Cartesian plane represent nearly anything in the real, physical world, they will not be completely collinear; small non-linearities will exist. Therefore, the curves will intersect at "almost" the same point.

1. Resolution Grids: The Duda and Hart Technique

One possible method for extracting these key intersection points in the Hough domain is to divide the Hough domain into grid squares. A count would be kept of the number of curves that pass through each square in the grid. The square with the most

curves passing through it must contain an intersection or group of intersections in a small neighborhood. The original line in the Cartesian plane can then be approximated by the (θ, ρ) pair at the center of the grid square. If more accuracy is required in the approximation, simply reduce the size of the grid square.

This is the concept behind the method proposed by Duda and Hart [Ref. 17]. Assuming it would be tedious and inefficient to analyze the Hough domain explicitly to find the precise intersections, Duda and Hart proposed dividing the Hough domain into a two-dimensional grid. The grid resolution would be based upon how much noise or 'scatter' existed in the Cartesian domain. Each cell in the grid represents a (θ, ρ) *region* where transformations of *almost* collinear points will *nearly* intersect. Each cell in the Hough domain is systematically analyzed to determine the set of curves that pass through it. Finally, the set of the corresponding coordinate points in the Cartesian plane must constitute an approximate line, approximately defined by the (θ, ρ) coordinates of the cell in the Hough domain. The general procedure is outlined in Table 1. Previous work at the Naval Postgraduate School on mobile robot localization employed this technique [Ref. 24].

Step 1.	For a given point, generate a θ-ρ curve plotted on the parameter plane grid.
Step 2.	Note the cells that the curve crosses.
Step 3.	Repeat Steps 1 and 2 for every point.
Step 4.	Count the number of crossings in each cell.
Step 5.	Recover the points whose curves contributed to the total of each cell.
Step 6.	Estimate a line for each set of points.

Table 1. Steps for Duda and Hart technique (After Ref. [24]).

Although this technique is quite popular, especially among image processing researchers, it is included in this thesis for information only. It will not be used in this thesis. Rather, intersections will be solved for explicitly and then clusters will be grouped by an unsupervised neural network.

2. Explicitly Solving for Intersections

It is desired to explicitly express the (θ, ρ) point where two curves intersect in terms of the original points in the Cartesian domain from which the curves were transformed.

As stated earlier, a point (x_o, y_o) in the Cartesian domain will be transformed in the Hough parameter domain to a sinusoidal curve. That curve was given earlier in Equation 4 as:

$$\rho_0 = x_0 \cos\theta + y_0 \sin\theta$$

Assume there are two curves ρ_1 and ρ_2 in the Hough domain which are the transforms of two points in the Cartesian domain (x_1, y_1) and (x_2, y_2) respectively. The curves are given by the transformation equations

$$\rho_1 = x_1 \cos\theta + y_1 \sin\theta$$

$$\rho_2 = x_2 \cos\theta + y_2 \sin\theta$$

It is a fundamental fact of geometry that any two points in the Cartesian domain are collinear, so it stands to reason that curves ρ_1 and ρ_2 must intersect. Let (θ, ρ) be the point in the Hough domain where the two curves intersect.

At the particular value of θ where the two curves intersect, we must have the condition that $\rho_1 = \rho_2$. By substituting the transformation equations, we have

$$\rho_1 = x_1 \cos\theta + y_1 \sin\theta = x_2 \cos\theta + y_2 \sin\theta = \rho_2$$

or, equivalently, $$(x_1 - x_2)\cos\theta + (y_1 - y_2)\sin\theta = 0.$$

Dividing both sides of the equation by the cosine of θ gives

$$(x_1 - x_2) + (y_1 - y_2)\left(\frac{\sin\theta}{\cos\theta}\right) = 0$$

Finally, collecting like terms and re-arranging, we arrive at an explicit value for the particular value of θ where the two curves intersect, in terms of the Cartesian coordinates:

$$\tan\theta = \frac{-(x_1 - x_2)}{(y_1 - y_2)} = \frac{(x_2 - x_1)}{(y_1 - y_2)}$$

$$\theta = \arctan\left(\frac{x_2 - x_1}{y_1 - y_2}\right), \text{ for } y_1 \neq y_2$$

or, equivalently, (5)

$$\theta = \frac{\pi}{2} \qquad \text{for } y_1 = y_2$$

Equation 5 gives an expression for θ which is a function only of the coordinates of the original points in the Cartesian domain. Once θ is known, Equation 4 can be used to solve for the corresponding ρ:

$$\rho = x_1 \cos\theta + y_1 \sin\theta = x_2 \cos\theta + y_2 \sin\theta \qquad (6)$$

Hence, given any pair of points (x_1, y_1) and (x_2, y_2) in the Cartesian domain, the precise locations of their intersections in the Hough parameter domain can be solved for by Equations 5 and 6. This is significant, because it implies that it is not necessary to analyze the *entire* curve in order to isolate the interesting points on the curve. This will substantially reduce processing time of the proposed algorithm.

It is important to note that the inverse tangent function will yield two possible values for θ, separated by π radians. Each θ will yield a different value for r when used in Equation 6; which differ by a factor of (-1). This implies that the Hough domain is symmetric; i.e., (x_1, y_1) and (x_2, y_2) will transform into curves which intersect at *two places* in the Hough domain. Due to the symmetry, however, knowledge of only one (θ, ρ) pair is sufficient.

If multiple coordinates are transformed, and their curve intersections solved for explicitly, it can be expected that multiple points in the Hough domain will be at the (θ, ρ) corresponding to a line in the Cartesian plane. In a real environment where noise exists, it can be expected that "clusters" of points will be congregated near the (θ, ρ) corresponding to a line in the Cartesian plane. If explicit solution is used, some means must be used to cluster these data points in the Hough domain and interpret a meaningful result. It also must be noted that the number of clusters in the Hough domain will not be known a priori. Clustering of data into an unspecified number of groups using neural networks is the subject of Chapter 5 of this thesis.

D. CHAPTER SUMMARY

In this chapter, the concept of the Hough transform was introduced. The *resolution grid* method for its implementation was presented and discussed briefly. A set of equations for finding the key intersections in the Hough domain explicitly were derived. Explicitly solving for intersections in the Hough domain will result in noisy groups of points when lines in the Cartesian plane are not perfectly collinear. This will almost certainly be the case for real sonar data. The following chapter presents a means for clustering these data and representing them with exemplar vectors.

V. NEURAL DATA CLUSTERING

In this chapter it will be shown how a neural network may be used to classify clusters of data in two dimensions. Although many methods exist to cluster data, a variation of the competitive "winner-take-all" neural network has been chosen for this application because of its simplicity and inherent resistance to noise. To begin with, an introduction to the "winner-take-all" competitive network is discussed. Next, that network is modified slightly from its traditional form to enable data representation in cases where the number of clusters is not known beforehand. Finally, the network is tested on some sample clusters in two dimensions and computation times are measured.

The proposed network is demonstrated in terms of a generic two dimensional pattern space. Later, in Chapter 6, the network will be applied in the specific two dimensional pattern space of the Hough domain.

A. "WINNER-TAKE-ALL" COMPETITIVE NETWORKS

Competitive networks are examples of *unsupervised learning* networks. Unsupervised learning implies that the network is presented a set of training data, but is not given a corresponding set of target outputs for each input. Rather, the network organizes the training patterns into classes on its own.

The "winner-take-all" learning rule, also referred to in some texts as the *Kohonen* learning rule [Ref. 30], differs from most other learning rules in that it cannot be explained or implemented in terms of a single neuron. Networks employing this learning rule will be an array, or *layer*, of neurons with massive interconnections as shown in Figure 8. The output nodes (neurons) compete, and the one with the maximum response is allowed to fire. When weights are updated, only the weights of the winning neuron will be changed; all others will remain the same.

Learning in this type of network is based on the clustering of input data to group similar inputs and separate dissimilar ones. *Similarity* in this case becomes synonymous with dot product; normalized vectors which are very similar will have a dot product of nearly one. Inputs in the pattern space $\Re^n$ are compared to (i.e. dotted with) p weight

vectors, also in $\Re^n$. The highest dot product wins the competition. Hence, the Kohonen network classifies input vectors into one of the p categories specified by the weight vectors.

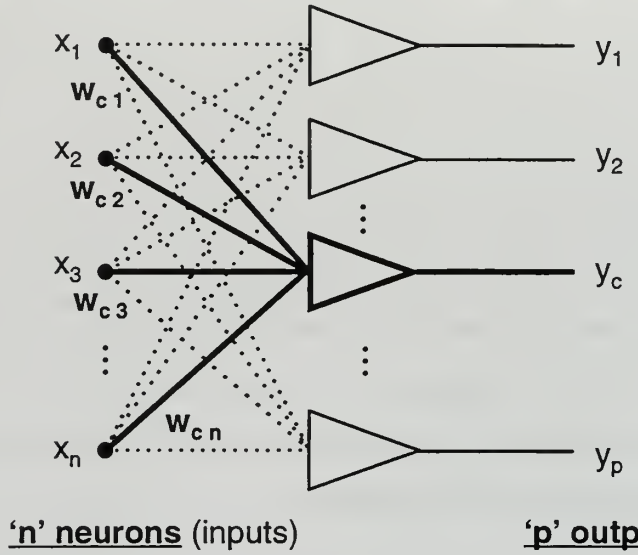

Figure 8. Kohonen, or "winner-take-all" network. (Highlighted weights are updated.)

Assume an input vector **x** in the pattern space $\Re^n$. Let **x** be a *normal* vector, i.e., a vector with length equal to 1. The first stage of the computation is the competition. A set of p random weight vectors normalized in $\Re^n$ is initially created, denoted by the weight matrix **W**. The output vector **y** is determined by the product of the weight matrix with the input vector **x**.

$$\underset{p\times 1}{\mathbf{y}} = \underset{p\times n}{\mathbf{W}} \bullet \underset{n\times 1}{\mathbf{x}} \qquad (7)$$

The largest element in **y** represents the output of the winning neuron, denoted y_c. Note that since the input vector and the weight vectors are normalized, y_c will have a maximum value of 1. The weights of the winning neuron (the c^{th} row of **W**) are then updated by

$$\hat{\mathbf{w}}_c^{new} = \mathbf{w}_c^{old} + \alpha(\mathbf{x} - \mathbf{w}_c^{old}) \qquad \text{where } 0 < \alpha < 1$$

and re-normalized.

$$\mathbf{w}_c^{new} = \frac{\hat{\mathbf{w}}_c^{new}}{\left\|\hat{\mathbf{w}}_c^{new}\right\|}$$

The process is then repeated for the next input vector **x**. The equations above may be generalized for the k^{th} iteration as a function of input vector $\mathbf{x}^k$, and are given in Equations 8, 9, and 10:

$$\underset{p\times 1}{\mathbf{y}^{k}} = \underset{p\times n}{\mathbf{W}^{k}} \bullet \underset{n\times 1}{\mathbf{x}^{k}} \tag{8}$$

$$\hat{\mathbf{w}}_c^{k+1} = \mathbf{w}_c^k + \alpha(\mathbf{x} - \mathbf{w}_c^k) \tag{9}$$

$$\mathbf{w}_c^{k+1} = \frac{\hat{\mathbf{w}}_c^{k+1}}{\left\|\hat{\mathbf{w}}_c^{k+1}\right\|} \tag{10}$$

The learning rate, α, is often chosen to be a constant. However it may, also be chosen to vary with k, depending on the designer's needs. Relatively high values of α will make the weights converge to the clusters they represent more quickly, but they will also cause "outlying" points to have a greater effect. Small values will make the network more resistant to noisy data, but the network will also take longer to converge.

After each of the input vectors has been presented to the network once, the first *epoch* has been completed. Each of the data will have been grouped into one of the p clusters, and the weight vectors representing those clusters will have moved toward those collective points. Further epochs may be necessary if those weights have not converged to an accurate representation of the points. Generally, some test is implemented to check whether the total change in weight values is small enough during an epoch and, if it is, the network is assumed to have converged.

B. NORMALIZATION PROCEDURE

As stated earlier, it is important that the input vectors, and the weight vectors, be normalized, else the dot product comparison in Equation 8 will be inconsistent and less meaningful. It is equally important that the original vector must be recoverable from the normalized vector. A particular strategy for accomplishing this normalization is outlined below. This procedure is modified somewhat from the procedure found in Reference 31.

The strategy for this normalization procedure is to represent data in n dimensions by equivalent normalized vectors in $(n+1)$ dimensions. The data in the original n

dimensions will each be scaled independently to make them approximately the same range, and the last dimension will be added in order to make the length of the new vector exactly equal to one.

Assume a data vector $\mathbf{V}$ in $\Re^n$ which represents the data to be input to the competitive network, but is not normalized.

$$\mathbf{V} = \begin{pmatrix} v_1 & v_2 & v_3 \ldots v_n \end{pmatrix}$$

If possible, scale each element in $\mathbf{V}$ by dividing it by a constant slightly larger than the maximum value it takes on in any of the vectors in the data set. For example, if v_1 represents an angle in radians, divide it by π. If v_2 represents a range finding from a sonar transducer, then divide it by the maximum detectable range of the transducer. Thus create a new vector $\mathbf{V}'$ whose elements are each less than or equal to one in magnitude

$$\mathbf{V}' = \left(\frac{v_1}{\max(v_1)} \quad \frac{v_2}{\max(v_2)} \quad \frac{v_3}{\max(v_3)} \quad \cdots \quad \frac{v_n}{\max(v_n)} \right) = \left(v_1', v_2', v_3' \ldots v_n' \right)$$

Second, choose a value N which is slightly greater than the maximum length of $\mathbf{V}'$. If the first step was conducted properly, then N will be the square root of n.

$$N = \sqrt{n}$$

Third, add a new entry d to the vector.

$$\mathbf{V}'' = \left(d, v_1', v_2', v_3' \ldots v_n' \right)$$

Fourth, set the new element d to a convenient value.

$$d = \sqrt{\left(N^2 - \|\mathbf{V}'\|^2 \right)}$$

Finally, divide the vector $\mathbf{V}''$ by N to get a new vector $\mathbf{V}'''$ whose length is identically 1.

$$\mathbf{V}''' = \frac{\mathbf{V}''}{N}$$

The value of d has been constructed to be exactly as long as necessary to make the length of the vector $\mathbf{V}'''$ in $\Re^{n+1}$ equal to one. This normalization procedure has the advantage that the original vector is easily derived from $\mathbf{V}'''$ by ignoring the element d,

and multiplying the remaining elements by N and by their respective maximum values. The five steps used for the normalization procedure are summarized in Table 2.

Step 1	Start with data vector $\mathbf{V} = \left(v_1 \quad v_2 \quad v_3 \ldots v_n\right)$.
Step 2	Independently scale each of the elements: $\mathbf{V}' = \left(\frac{v_1}{\max(v_1)} \quad \frac{v_2}{\max(v_2)} \quad \frac{v_3}{\max(v_3)} \quad \cdots \quad \frac{v_n}{\max(v_n)}\right) = \left(v_1', v_2', v_3' \ldots v_n'\right)$
Step 3	Choose a constant equal to the maximum length; $N = \sqrt{n}$
Step 4	Add an element: $\mathbf{V}'' = \left(d, v_1', v_2', v_3' \ldots v_n'\right)$ where $d = \sqrt{\left(N^2 - \lVert \mathbf{V}' \rVert^2\right)}$
Step 5	Divide the new vector by the maximum length; $\mathbf{V}''' = \frac{\mathbf{V}''}{N}$

Table 2. Steps for normalization of data vectors

C. ADDING A CONTROL MECHANISM TO THE NETWORK

One of the primary drawbacks of the competitive network outlined above is that it classifies inputs into one of p outputs, where p is the number of neurons in the competitive layer. Hence, it is necessary to know beforehand the number of clusters the network is looking for. This burdensome requirement can be alleviated by adding a control mechanism to dynamically adjust p after each epoch.

The first epoch should be conducted with a value of p that is much higher than the expected number of clusters in the pattern space. At the end of the epoch, a series of tests are conducted:

1. If any neuron has weight values identical to the weights it had at the beginning of the epoch, then none of the data were classified by it. The neuron is eliminated.
2. If any two weight vectors are *similar*, i.e. if their dot product exceeds some maximum value NN_{TOL}, then the two neurons are combined and a new random weight vector is created.

3. (Test for convergence) If no weight vectors were eliminated and no weight vectors were combined during the last two consecutive epochs, then assume that the network has converged. Otherwise, conduct another epoch.

Without the modification of this added control mechanism, this particular network is commonly referred to as a *competitive* network. Taking the competition one step further, these neurons now compete *for survival*. Losing neurons are eliminated, and in the end only those neurons which consistently won competitions survive. In this sense, the network might be called a "Survival of the Fittest" network.

D. DATA CLUSTERING USING THE CUSTOMIZED NETWORK

The Kohonen network described in this chapter, along with the modifications described regarding normalization and control of the parameter p, were implemented and included as Appendix B. The four coordinates $\{(0.2,0.2),(0.2,0.8),(0.8,0.2),(0.8,0.8)\}$ were used to create the clusters of data shown in Figure 9.

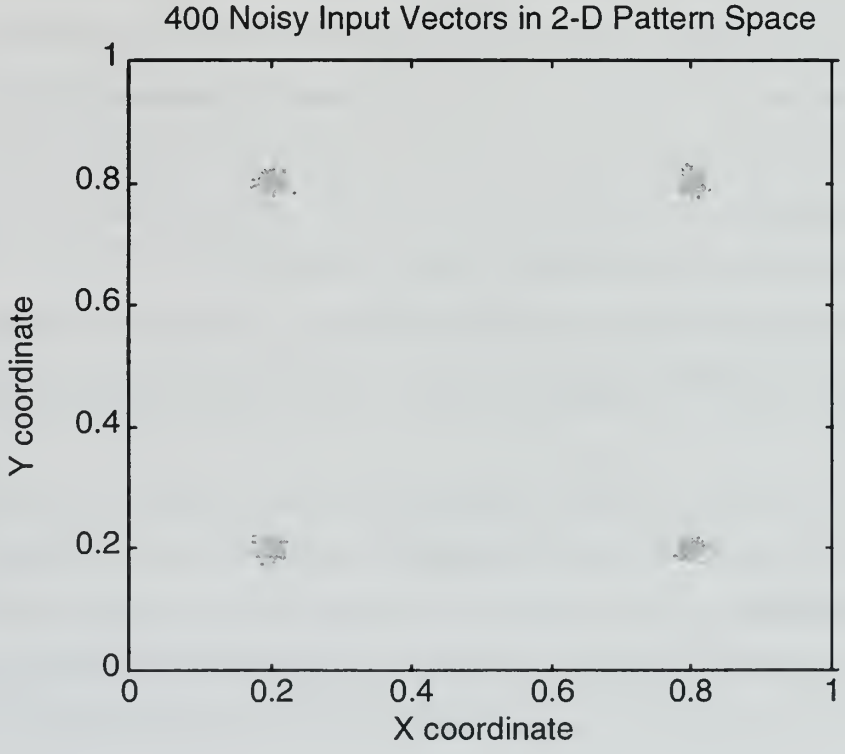

Figure 9. Test vectors for neural network clustering

A human can easily inspect Figure 9 and come to the conclusion that there are exactly 4 clusters of data, approximately given by the coordinates:

$$\{(0.2,0.2),(0.2,0.8),(0.8,0.2),(0.8,0.8)\}$$

The challenge is to devise a network that can perform these tasks without human assistance. The objective will be for the network to determine that there are exactly 4 clusters of data, and that these clusters are represented by exemplar vectors which are reasonably close to these coordinates.

The points were presented to the network, and an initial value of $p = 30$ was chosen. Figure 10 shows the initial placement of the random weight vectors in the 2 dimensional representation. Recall that the true weights are actually 3 dimensional due to the normalization process. What is plotted is actually the two-dimensional representation of these weights *prior to* the normalization procedure.

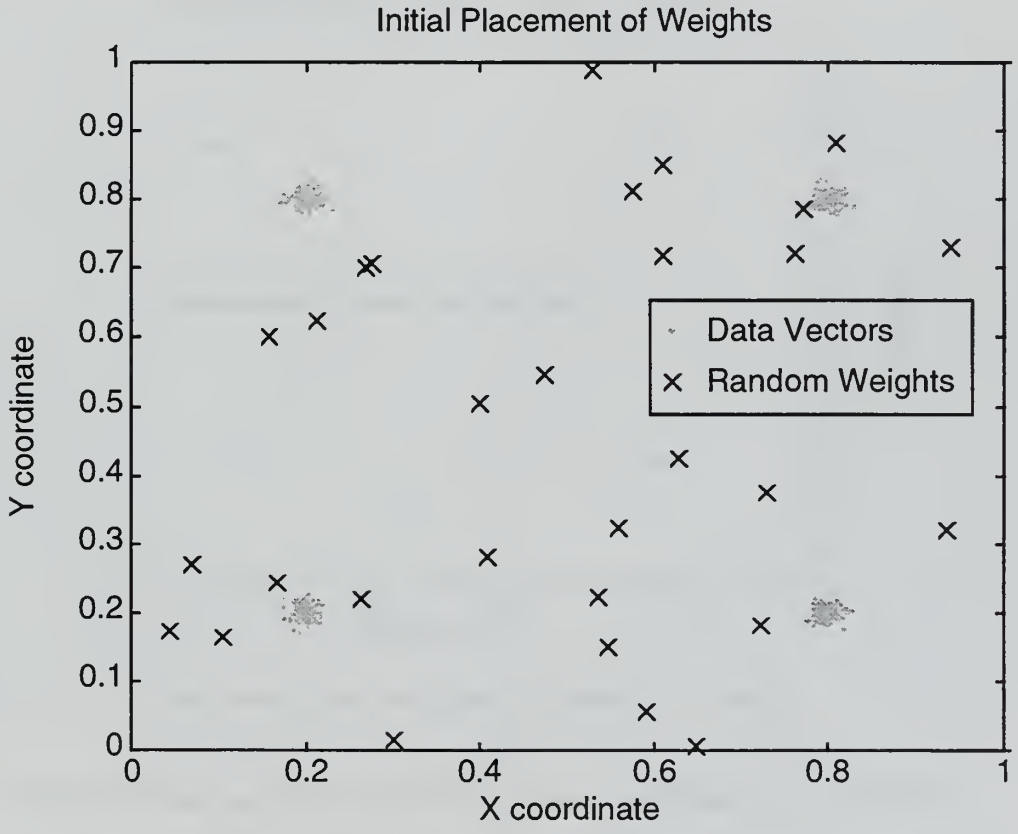

Figure 10. Initial placement of random weights

The data vectors were presented to the network, as implemented in Appendix B, and a learning rate of $\alpha = 0.5$ was defined. A tolerance of $NN_{TOL} = 0.98$ was defined for

the dot product similarity test of weight vectors. In a reasonable time, the network was able to determine that there were in fact 4 clusters of data, and converged to represent these 4 clusters by the 4 exemplar vectors shown in Table 3. As can be seen, these exemplar vectors are reasonably close to the coordinates from which these noisy data samples were derived.

X coordinate	0.2044	0.7940	0.2036	0.7954
Y coordinate	0.7982	0.2028	0.2099	0.7962

Table 3. Exemplar vectors chosen by neural network

The exemplar vectors chosen by the network are illustrated in Figure 11. When compared with the original data in Figure 9, the results appear to be quite consistent with what a human might have done.

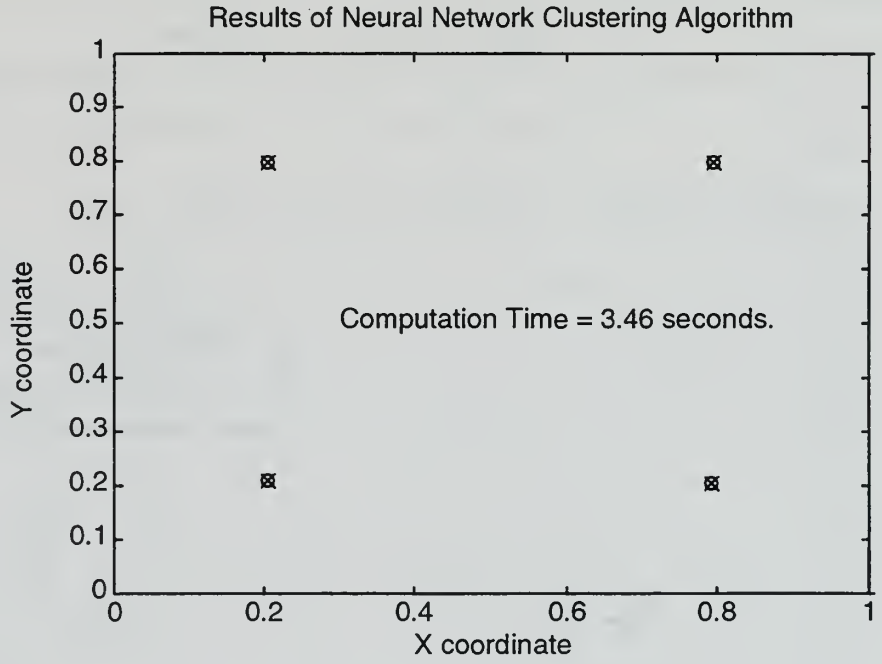

Figure 11. Results of neural network clustering

The network produced these exemplar vectors in 3.46 seconds. While the network is obviously not as fast as a human, these results are acceptable for the application chosen of finding clusters in the Hough domain in order to determine the location and orientation of walls near the robot.

E. CHAPTER SUMMARY

In this chapter the very common "winner-take-all" or "Kohonen" neural network was introduced. It was shown that this network can be used to cluster data, but is only useful when the number of clusters is known a priori. Modifications were made to this network to allow the number of neurons to vary from one epoch to the next, by eliminating neurons which do not win competitions. The result is an unsupervised network which clusters data without a priori knowledge of the number of clusters.

The following chapter will draw on the tools introduced in this chapter and Chapter IV. Specifically, the way in which the issue of localization might be addressed using these tools is outlined in detail.

VI. IMPLEMENTATION

This chapter serves to tie together concepts covered earlier in this thesis, discussing the fashion in which the Hough transform (covered in Chapter IV), and the neural network (covered in Chapter V) may be used to process sonar data from the robotic system (covered in Chapter II) to solve the problem of localization (covered in Chapter III). As outlined earlier, the proposed solution consists of 4 basic parts:

1) The range data returned from the Sensus 200 system are converted to (x, y) pairs representing the Cartesian points (in robot coordinates) where the echo occurred.
2) These (x, y) coordinates will be parameterized using Hough parameters.
3) Regions in the Hough domain where curves tend to intersect each other must be represented by clusters of points.
4) A competitive neural network will be employed to identify clusters and represent them by a single point.

The implementation of these steps was done in three function programs written for MATLAB, and will be covered in detail in this chapter. Steps 2 and 3 will be combined into a single function by selectively and explicitly solving for intersections in the Hough domain.

The three programs should be taken as an overall system as illustrated in Figure 12, The inputs consist of range data from the sonar system, and the outputs are (θ, ρ) pairs representing the distance and orientation of nearby walls.

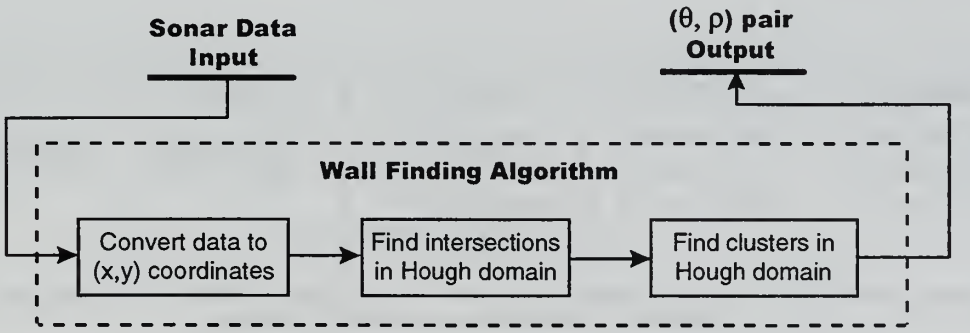

Figure 12. Overall wall finding algorithm

A. CONVERTING SONAR DATA TO X-Y COORDINATES

The code used to gather range data with the Nomad Scout is included in Appendix D. Originally, the code was intended to also be used with other Nomadic robots, so a 5-column output was desired. Column 4 is a function of the "Turret Angle" and is not used with the Nomad Scout, but is used with other Nomadic robots at the Naval Postgraduate School.

The robot was programmed to cycle through each of the 16 sonar transducers, then rotate 7.5° counter-clockwise and cycle through each of the 16 transducers again, then rotate 7.5° counter-clockwise and cycle through each of the 16 transducers a third time. This results (ideally) in 48 range findings equally dispersed about the robot.

As noted earlier, the robot is able to track its dead reckoning position and orientation. At startup, the robot marks its current position as the origin and the direction of sonar (1) as the x axis. This becomes the *world coordinate frame*. As it navigates, it tracks its position relative to these initial settings. Since it was desired to process the data off-line in order to demonstrate the concept, data were written to an ASCII file in the five-column format shown in Table 4. The net result will be a matrix of data with 48 rows (the total number of sonar range findings gathered) and 5 columns. The first 16 rows will be the range findings of sonar transducers 1 through 16 (in that order) at the initial orientation. The next 16 rows will be the range findings at the 7.5° counter-clockwise offset, and the final 16 rows will be the range findings at a 15° counter-clockwise offset.

Column 1	Column 2	Column 3	Column 3	Column 5
(Dead Reckoning X Coordinate in inches) times 10	(Dead Reckoning Y Coordinate in inches) times 10	(Dead reckoning orientation of robot relative to X-axis in degrees) times 10	Not used with Scout robot.	Range return of i^{th} sonar transducer in inches

Table 4. Data format of range findings

Once the data are gathered and collected in this matrix form, it is a fairly straightforward mathematical process to represent the range findings as (x, y) pairs in the

world coordinate system relative to the origin and x-axis defined at startup. The implementation is included in Appendix C.

First, unreliable data in the matrix must be discarded. The maximum range finding of the sonar transducers is 255 inches. Among researchers at the Naval Postgraduate School, however, experience has shown that data can be unreliable even at much smaller ranges. For this application, we choose to rely only on range findings that are less than 110 inches. Any row whose entry in column 5 exceeds 110 is simply discarded. Similarly, range data less than 17 inches are also considered unreliable. Any row whose entry in column 5 is less than 17 is discarded.

Next, for each range finding, the robot must know its own (x, y) position in the world coordinate system. This will be the location of the center of the robot, relative to the startup origin, as tracked by dead-reckoning. This particular application envisions the robot taking these readings at or near the world coordinate origin. Other applications would require minor revisions to the code in the appendices. Units are chosen to be inches. Ideally, since the robot is simply rotating to take the 48 returns, these two values should be identical for all 48 returns. In reality, however, each set of 16 returns will be taken from a slightly different (x, y) position. This is due to the fact that the Scout is not a truly holonomic system; its wheels must move in order for the robot to rotate.

$$X_{robot} = \frac{Column1}{10} \tag{11}$$

$$Y_{robot} = \frac{Column2}{10} \tag{12}$$

Third, the robot's orientation for each range finding must be known. This value should be identical for each set of 16 returns, since the robot is stationary when these returns are taken. Units are chosen to be radians.

$$\theta_{robot} = \left(\frac{Column3}{10}\right)\left(\frac{\pi}{180}\right) \tag{13}$$

It follows from geometry that the x coordinate of the range finding should be the x coordinate of the robot, plus the quantity of the range times the cosine of the angle of the

range finding. It is important to include the angular offset of each transducer in the computation. The angle between transducers is 22.5°, as shown in Figure 13.

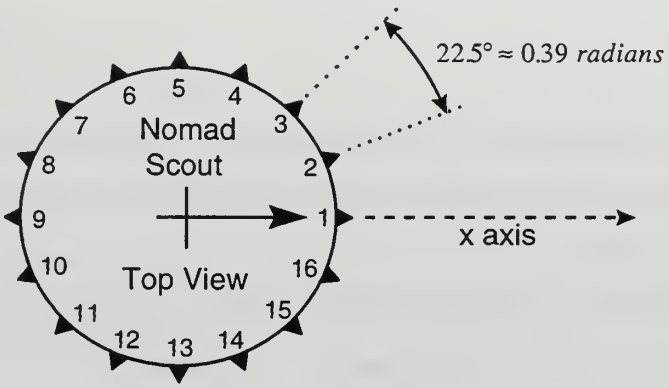

Figure 13. Angular relationship of transducers

It is also important to remember that the range finding is relative to the transducer, *not* the center of the robot. Hence, the radius of the robot must be added to this value. This value was measured in the laboratory to be approximately 7.2 inches. The y coordinate can be similarly calculated. The (x, y) location of the sonar return from a transducer is given by Equations (14) and (15):

$$X_{return} = X_{robot} + \left(\frac{Column5}{10} + R\right)\cos\left(\theta_{robot} + \frac{(22.5)\pi}{180}(j)\right) \tag{14}$$

$$Y_{return} = Y_{robot} + \left(\frac{Column5}{10} + R\right)\sin\left(\theta_{robot} + \frac{(22.5)\pi}{180}(j)\right) \tag{15}$$

where, R is the radius of the robot (7.2 inches for a Nomad Scout) and j is the index of the individual transducer $(1 \le j \le 16)$.

As a final step, the resulting data points are sorted in counterclockwise order starting with the point closest to the x-axis (see Figure 14).

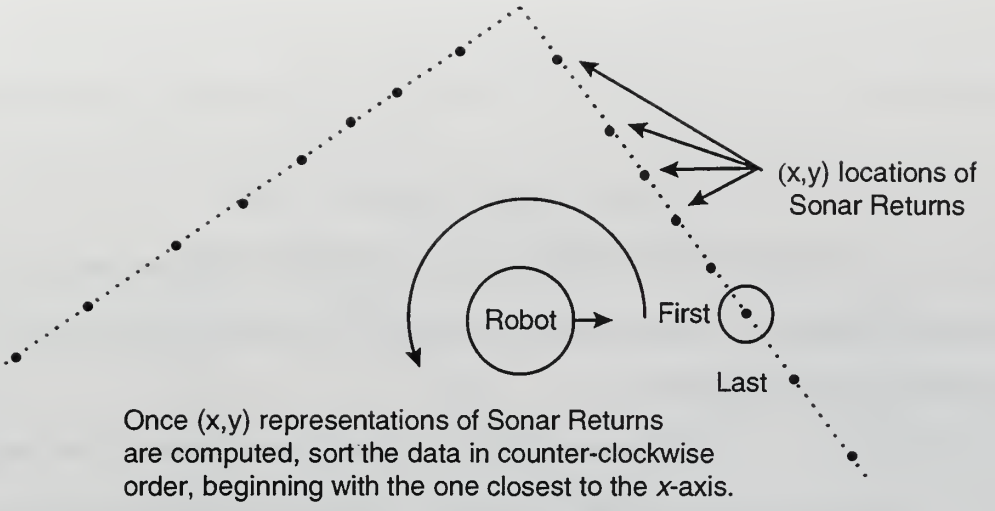

Figure 14. Cartesian data are sorted counter-clockwise

B. REDUCED HOUGH TRANSFORM OF SONAR RETURNS

Each of the (x, y) pairs resulting from the conversion of the range data can be transformed under the Hough parameters to a curve. However, it is not necessary to know the entire curve in order to find walls near the robot. Walls in the world (Cartesian) coordinate system will be transformed to points in the Hough domain; specifically, points where curves intersect. Hence, it is only necessary to find those points in the Hough domain where the curves representing the transformed sonar data intersect.

Further, it is not possible that the return from transducer 1, for example, will return an echo from the same straight-line wall as transducer 9. These transducers are facing in opposite directions. Hence, we can reduce the time required to find intersections in the Hough domain by checking each curve for intersections with the transformed curves of its four closest neighbors; two clockwise and two counter-clockwise. These four intersection points will only be included if they are all within a reasonably small neighborhood of one another. The implementation in MATLAB version 4.2 (b) of this reduced representation in the Hough domain is included with this thesis as Appendix A.

In Chapter IV it was shown that for any two points in the Cartesian domain, the intersection of their curves in the Hough domain was given by Equations 5 and 6:

$$\theta = \arctan\left(\frac{x_2 - x_1}{y_1 - y_2}\right), \text{ for } y_1 \neq y_2 \qquad (5)$$
$$\theta = \pi \qquad \text{for } y_1 = y_2$$

$$\rho = x_1 \cos\theta + y_1 \sin\theta = x_2 \cos\theta + y_2 \sin\theta \qquad (6)$$

Once the (x, y) coordinates of the sonar returns are determined, and these data are sorted as illustrated in Figure 14, we apply the following steps to determine clusters of key intersections in the Hough domain:

1) For each (x, y) point, find the (θ, ρ) locations in the Hough domain where its transformed curve intersects the curves of its 2 nearest clockwise and 2 nearest counterclockwise neighbors.
2) Check to see if these four (θ, ρ) points are within a reasonably small neighborhood of one another. Appropriate values to select for this test will be covered in the Chapter 7.
3) If they are, then "tag" all four (θ, ρ) points for presentation to the clustering algorithm, and move on to the next (x, y) point. If not, then simply move on to the next (x, y) point.
4) Finally, note the symmetry in the Hough domain. Include only (θ, ρ) points where ρ is positive. In fact, only points where ρ is greater that the radius of the robot plus the minimum trusted range of the transducers need to be included.

For example, assume the points in the Cartesian domain are arranged as shown in Figure 15. When transformed to the Hough domain, these points become the curves shown in Figure 16. The reduced Hough transform of these points become groups of points near the key intersections, as shown in Figure 17. Since there is no noise in this case, the points are grouped very tightly around the key intersections; there are actually 56 points.

Since this process differs markedly from the conventional "Hough transform," it will be called the "Reduced Hough transform" in this thesis. The (θ, ρ) domain where these clusters of points exist will be referred to as the "Reduced Hough domain."

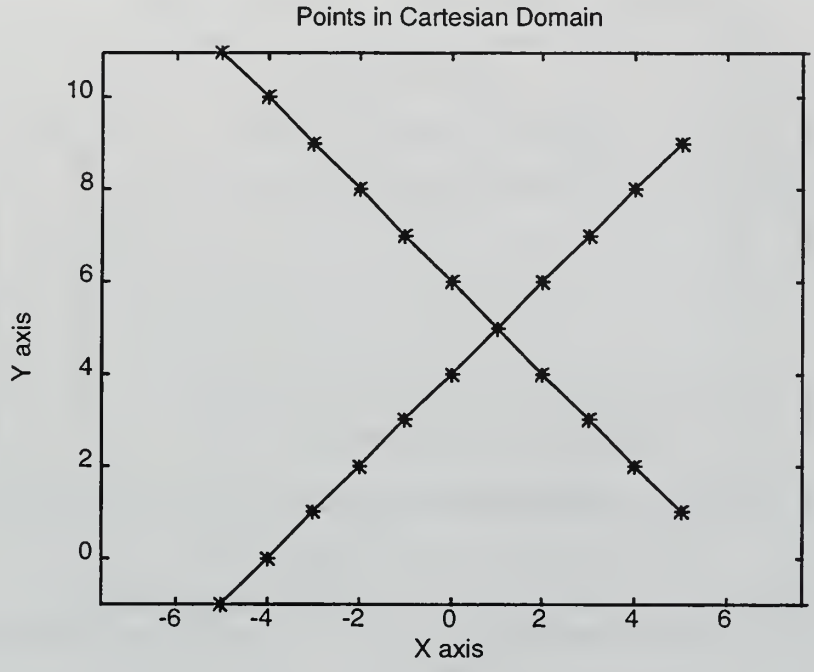

Figure 15. Test points in the Cartesian domain

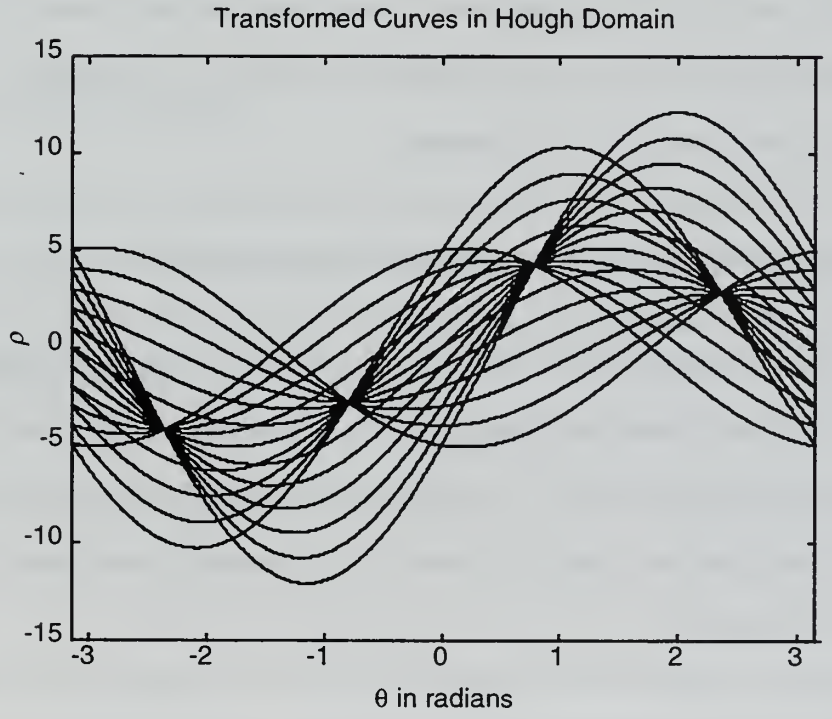

Figure 16. Test points transformed to curves in the Hough domain

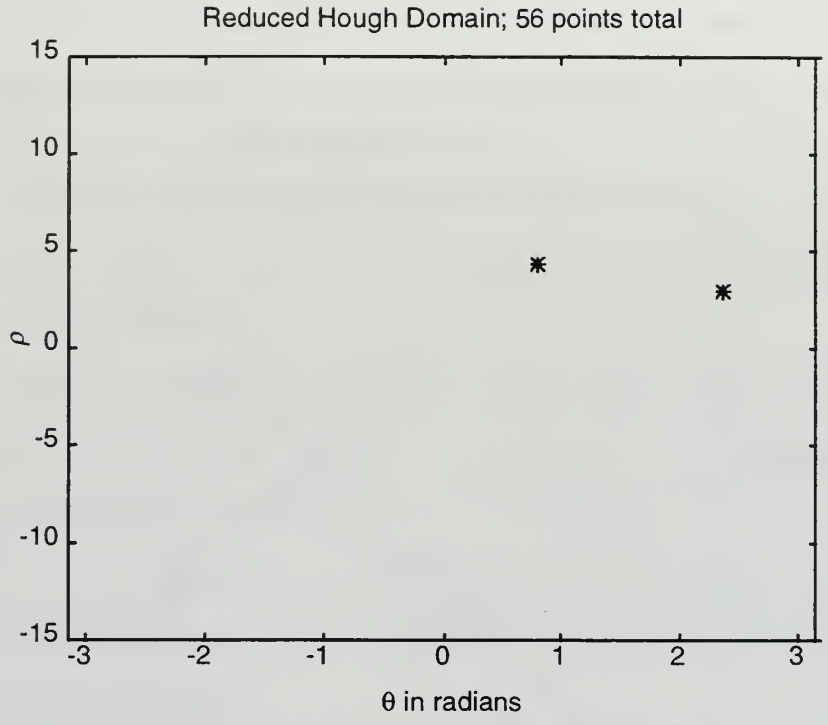

Figure 17. Test points transformed to clusters in the reduced Hough domain

At the conclusion of this portion of the algorithm, the result should be clusters of isolated points in the Hough domain, concentrated in regions where multiple (θ, ρ) curves tend to come very close to intersecting.

C. FINDING CLUSTERS IN THE HOUGH DOMAIN

The "Survival of the Fittest" network presented in Chapter 5 is the method chosen to group the data into clusters and represent them by exemplar vectors. Each (θ, ρ) point is treated as an input vector in $\Re^2$. During the normalization process, θ is divided by its maximum value (π radians), and ρ is divided by the maximum trusted range of the transducers (110 inches).

In Figure 18, the network was presented with the 56 points from the previous example. The number of neurons, p, was initially set to a value of 30. The figure shows

that the network found exactly 2 clusters of points. Further, it represented these two clusters with reasonably accurate exemplar vectors.

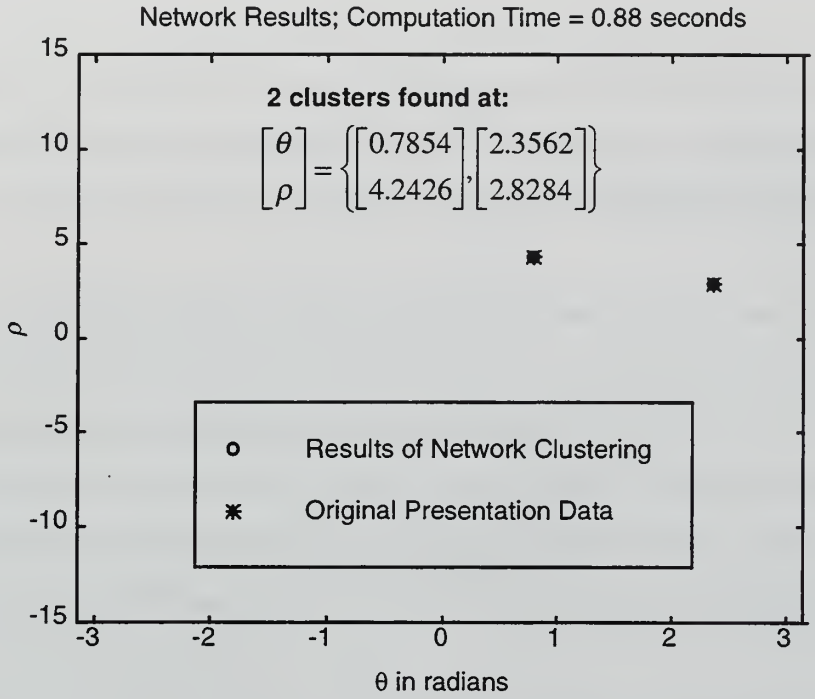

Figure 18. Results of network clustering in the reduced Hough domain

D. CHAPTER SUMMARY

In this chapter, the proposed algorithm for feature based localization in a mobile robot was discussed in detail, drawing on concepts introduced in earlier chapters. In the next chapter, the results of this algorithm applied in both simulated and actual indoor environments are presented.

VII. EXPERIMENTAL RESULTS

In this chapter the proposed algorithm (which is described in Chapter 6) is applied to sets of data and quantitative results are measured. Initially, simple tests are performed to determine whether the algorithm performs suitably under ideal conditions. Further modeling is then conducted with a simulated robot in order to demonstrate compatibility of the algorithm with the Scout platform. Finally, an actual Scout is used to take sonar readings from a real world indoor environment, and those readings are analyzed to find walls.

If the algorithm is to prove useful for the task of localization in an autonomous mobile robot, then the output(s) of the algorithm must be consistent for range data taken at the same location. The output need not be an accurate representation of the nearby walls (although this will be helpful if future research applies this algorithm to other tasks). It is necessary that the algorithm consistently identify the same number of walls, and at the same range and orientation, when presented with range data gathered at the same location. Consistency must be within:

- 1 inch (excellent) to 3 inches (adequate) for range to the wall, and
- 1 degree (excellent) to 3 degrees (adequate) for orientation of the wall.

Additionally, the algorithm must produce results in a reasonable amount of time. For this application, we consider 5 seconds to be an adequate goal. If the network is able to produce consistent results in less than 5 seconds, then it is suitable for localization of the mobile robot.

A. SIMPLE TESTS

Two simple tests were conducted to determine whether the algorithm was performing as expected under ideal conditions. The first of these simulates sonar findings under noise-free conditions. The second is intended to demonstrate that the algorithm does not require nearby walls to be perpendicular in order to find them. Walls may be placed at arbitrary angles to one another, and the performance of the algorithm is independent of these angles. The neural network learning rate was set to 0.5 for these

tests, and the Hough domain tolerances were set to 10 degrees and 10 inches. The relevance of these parameters will be discussed later in this Chapter. We do not yet test for consistency in the results, as these are simulated returns and free of noise. Rather, these two tests are simply intended to verify that the algorithm behaves in the anticipated manner.

1. A Simulated Corner

For the initial testing, an artificial 48 by 5 return matrix was devised which would simulate the returns of a nearby corner under ideal conditions. The matrix represents those returns that would arise from the setup shown in Figure 19, under absolutely ideal conditions (with no noise whatsoever). The robot is envisioned to be 50 inches from one wall, and 40 inches from another. The orientations of these walls are -45° and -135°, respectively. The robot is envisioned to perform 3 cycles of readings so that 48 returns will result; each return separated by 7.5°. Under noiseless conditions, the value of the returns was computed, and a matrix was created from these returns to imitate the output generated by the Scout executing the program in Appendix D.

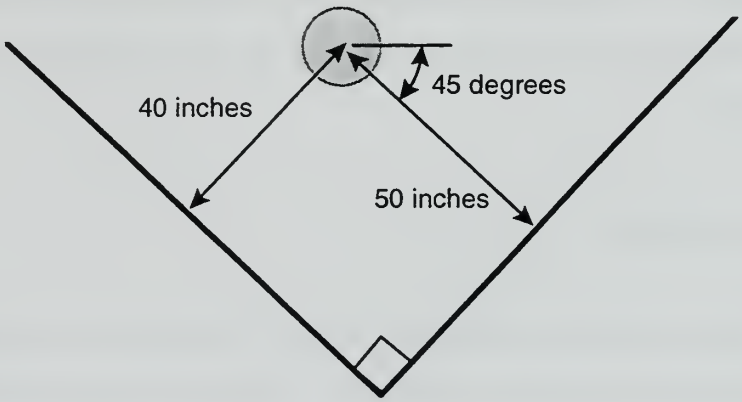

Figure 19. Simple test # 1: A simulated corner

The simulated returns are plotted in two dimensions in Figure 20, and the returns which fall within the trusted range (less than 110 inches but more than 17 inches) have

been circled. Note that the robot's perspective of its environment is only two dimensional.

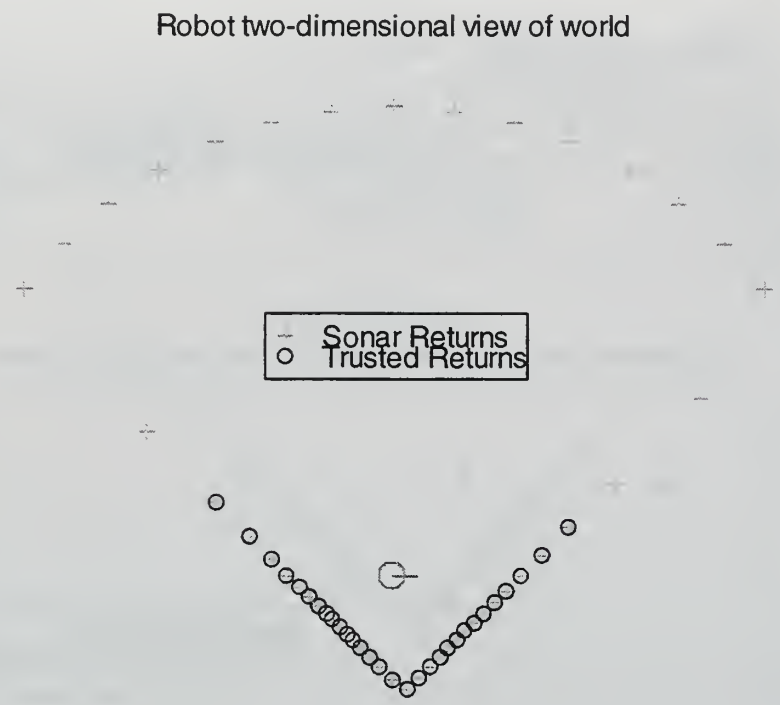

Figure 20. Simple test # 1: Robot's view of the world

The returns were converted to (x, y) points, and these points then transformed to the Hough domain. For the sake of clarity, the complete Hough transform of these points is shown in Figure 21 (a), although this was not used. The collection of intersection points shown in Figure 21 (b) was presented to the neural network for classification. The network was able to determine exactly two clusters of data, and represented those clusters by the exemplar vectors shown in Table 5.

θ	-45.0000	-134.9999
ρ	49.9999	40.0000

Table 5. Simple test # 1: Exemplar vectors

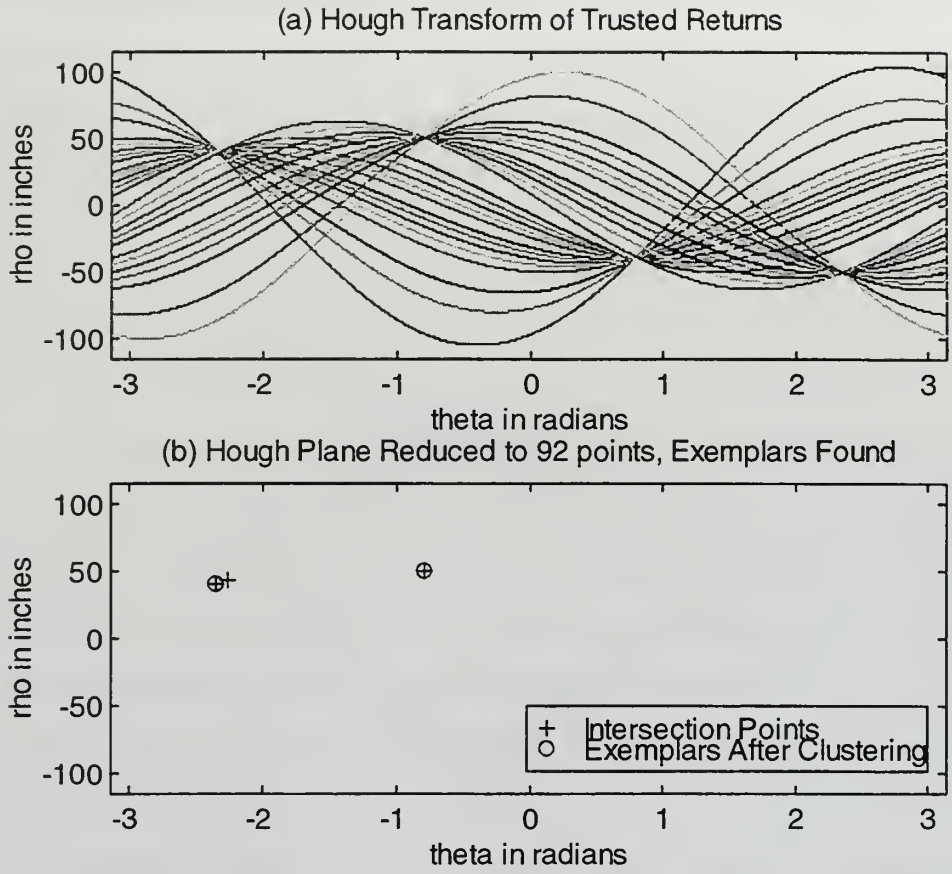

Figure 21. Simple test # 1: Hough domain representation of simulated sonar returns (a) Hough transform of all trusted returns (b) Reduced Hough transform and exemplars found during clustering.

If the algorithm is performing properly, then the exemplar vectors chosen by the network (see Table 5) should be reasonably accurate descriptions of the nearby walls. Figure 22 shows the walls chosen by the network superimposed over the original sonar returns. By inspection, the chosen vectors seem to coincide with the simulated walls. Additionally, the time required for the network to determine the range and orientation of these two walls falls well within the standard established of 5 seconds. From this test it appears that the algorithm is behaving as expected, and further investigation is warranted.

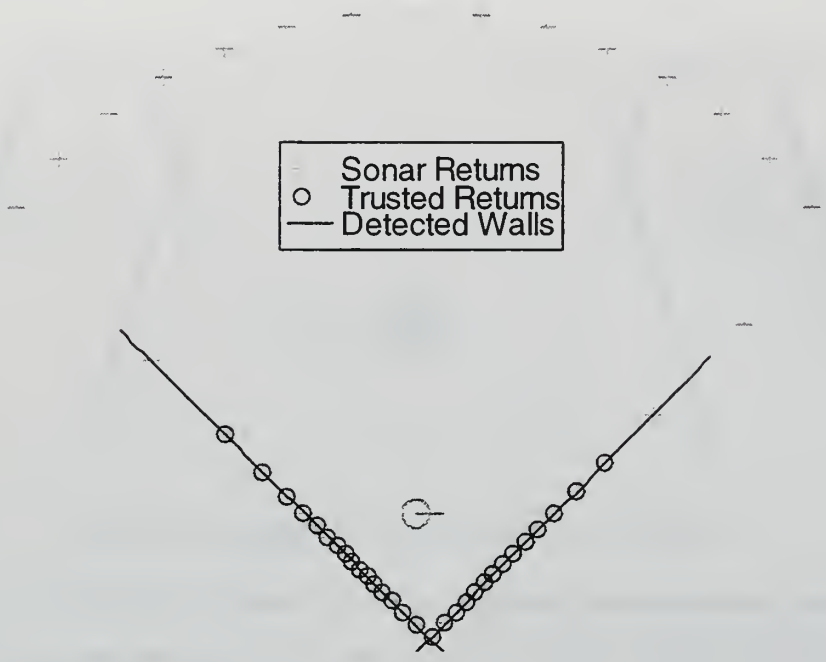

Figure 22. Simple test # 1: Simulated sonar returns and detected walls

2. A Case In Which Walls Are Not Orthogonal

The proposed algorithm has the property that it is independent of the orientation of the walls with respect to one another. Walls may be at arbitrary angles to one another, and the performance of the algorithm is independent of these angles. To demonstrate this a second test is performed, in which the walls near the robot are not perpendicular to one another. The scenario is constructed as shown in Figure 23, with the walls given by the ranges and orientations shown.

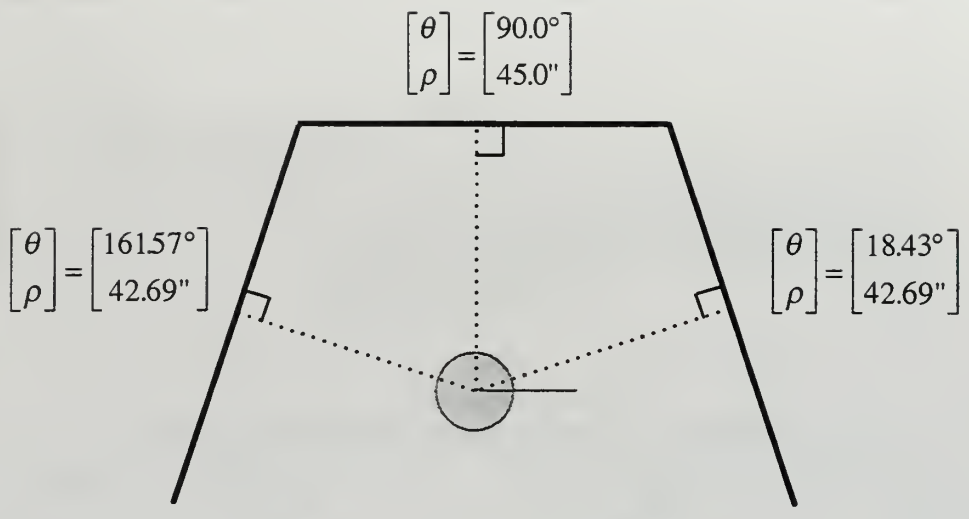

Figure 23. Simple test # 2: Non-orthogonal walls

Again, these data were simulated. A 48 by 5 matrix was designed to imitate the returns that would have occurred if the robot were in this environment, and ideal operating conditions were present so that sonar returns were free of noise (See Figure 24 (a)). The data were then presented to the algorithm, and exactly three walls were detected and represented by the exemplar vectors shown in Table 6. When compared to the true locations of these simulated walls given in Figure 23, the results appear to be acceptable even when the walls are not perpendicular.

θ	18.4348	89.9998	161.5652
ρ	43.1655	45.0000	43.1656

Table 6. Simple test # 2: Exemplar vectors

These results are illustrated in Figure 24. The detected walls are shown superimposed on the sonar data which form the robot's two-dimensional view of the world. The walls seem to have been placed in approximately the same place where a human viewing the data would have placed them. The network was able to accomplish this in less than two seconds, which is acceptable for this application.

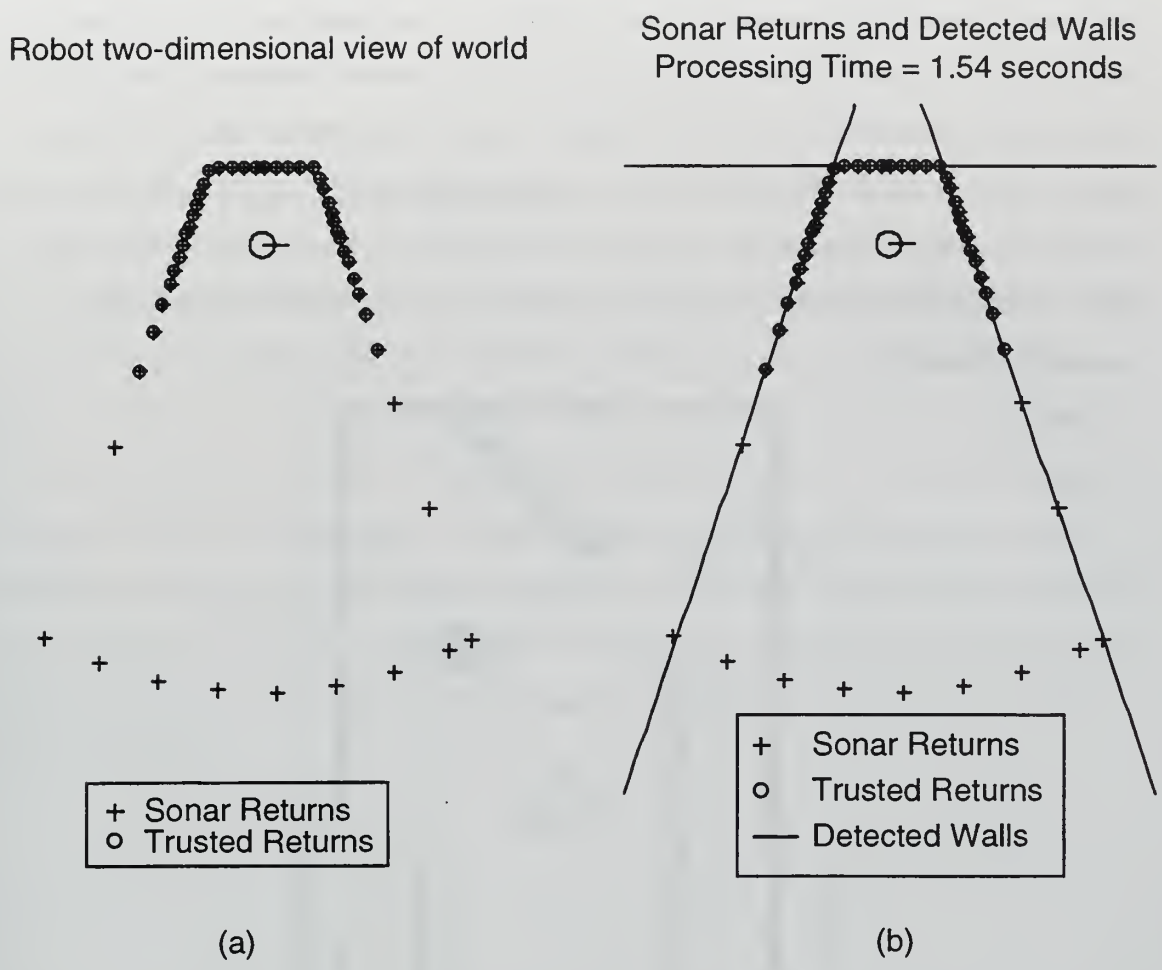

Figure 24. Simple test # 2: Simulated sonar returns and detected walls (a) Robot's two dimensional view of the world. (b) Detected walls superimposed on sonar range data.

This scenario demonstrates that nearby walls need not be orthogonal for the algorithm to function properly. In this scenario, walls which intersected at arbitrary angles were properly identified, and subsequently represented by accurate exemplar vectors. It also further supports the conclusion that the algorithm appears to be functioning properly.

B. SIMULATED ROBOT TESTING

Nomadic Technologies has developed a program called *NSERVER*™, which can be used to simulate its robots for the purpose of testing and developing programs. The program allows the designer to build complex environments easily, and simulate the

behavior of the robot in these environments. While the program does not completely simulate the non-ideal acoustic properties that affect the sonar transducers, the program does enable the algorithm to be tested in more complex scenarios in order to find cases where it may not work. Additionally, these tests verify the code used to gather the sonar data prior to implementation on the Scout. The program used to gather sonar data was run in several locations in the virtual environment shown in Figure 25, and the data analyzed to find walls.

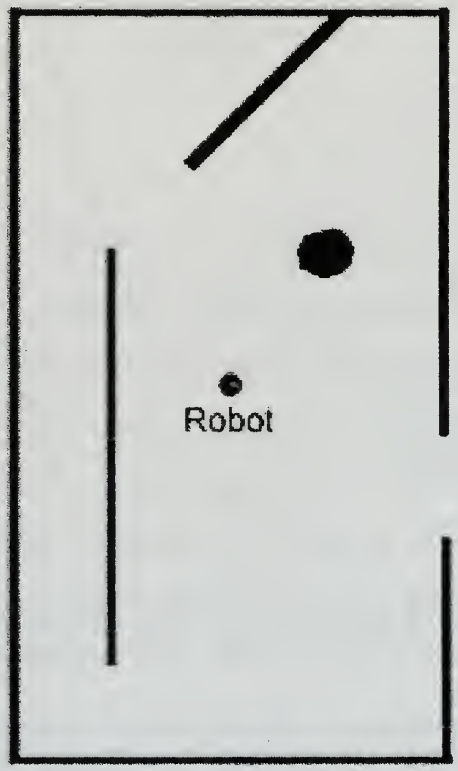

Figure 25. Virtual environment used for simulations

All of the simulation scenarios run in this virtual environment were conducted by the network using a constant learning rate of 0.5 in the neural network, and Hough domain tolerances of 10 degrees and 10 inches. These parameters are not ideal for noisy environments; adjustment of these and other parameters to optimize performance of the algorithm will be discussed later in this chapter.

Again, these data are noise-free. The results are not yet analyzed for consistency. Rather, we conduct these simulations to determine if there are scenarios in which the algorithm is unable to determine the correct number of walls, or produces grossly

inaccurate exemplar vectors. Also, we compare the detected walls to the sonar returns to verify that they generally coincide.

1. Corner of a Virtual Room

The first set of data was gathered after placing the virtual robot in the lower right corner of the map, facing generally toward the doorway, as shown in Figure 26. The direction the robot is facing is denoted by a white tick mark on the robot. Once placed, the program is able to provide the x and y positions of the robot on the map coordinate system, as well as the turret angle. Since the Nserver simulation program runs in *map coordinates*, and for this application it is desired to receive the data in *robot coordinates*, these values were simply noted at the time the robot was placed and subtracted from the appropriate columns in the 48 by 5 matrix before presentation of the data to the wall-finding algorithm.

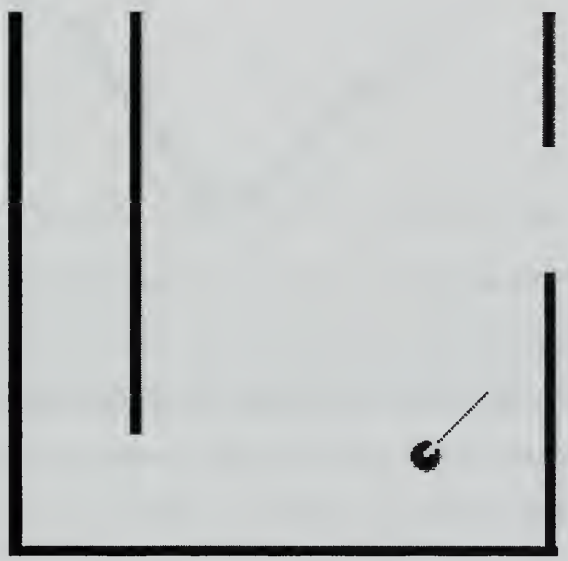

Figure 26. Simulation # 1: Problem setup

Once the data were collected, and adjusted to make the robot's position and orientation the origin and x axis, the matrix was presented to the algorithm and computation time was measured. In less than one second, the network determined that there were exactly two walls in the vicinity. It chose two exemplar vectors to represent

these walls, which are given in Table 7. Figure 27 shows the detected walls superimposed over the sonar returns. The detected walls differ by nearly 90 degrees, an indicator that the results are fairly accurate. Additionally, the walls appear to coincide with the sonar returns within an acceptable margin. The results from this first location would seem to indicate that the algorithm can handle this scenario reasonably well.

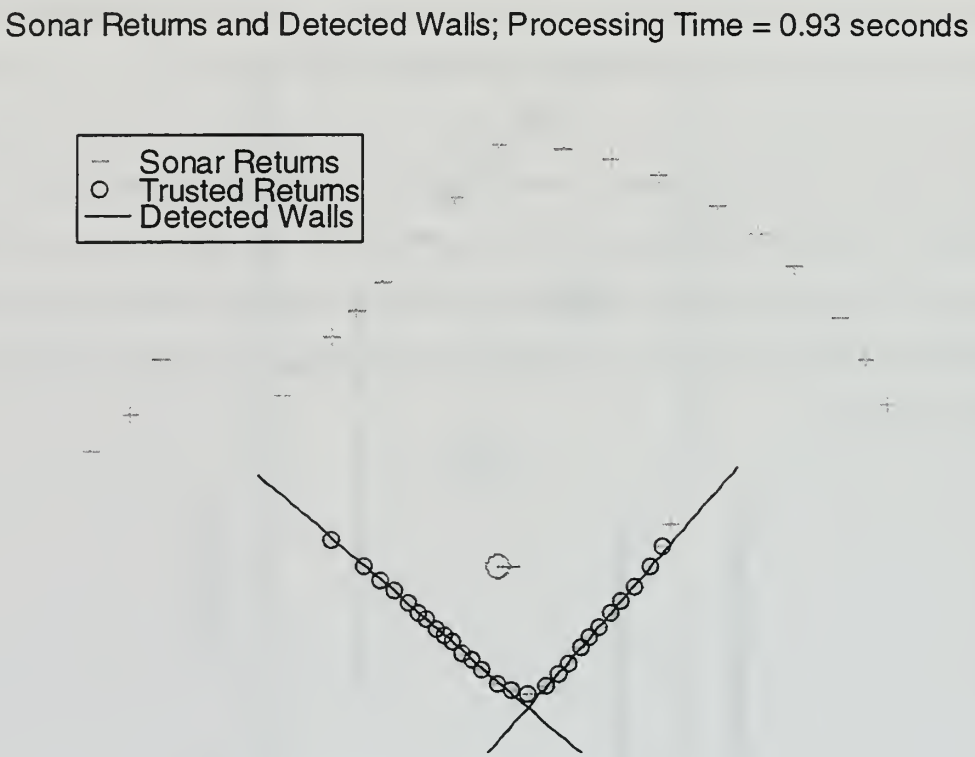

Figure 27. Simulation # 1: Sonar returns and detected walls

θ	-130.6130	-41.3164
ρ	54.4090	71.9664

Table 7. Simulation # 1: Exemplar vectors

2. A Corridor

In the second test, the robot was placed in a corridor as shown in Figure 28. The sonar range findings were processed in the same manner as the previous scenario. In this case, we expect the network to choose exemplar vectors which are nearly parallel to one another. We also expect to find that the walls, when superimposed on the sonar range findings, will coincide with the plotted range findings.

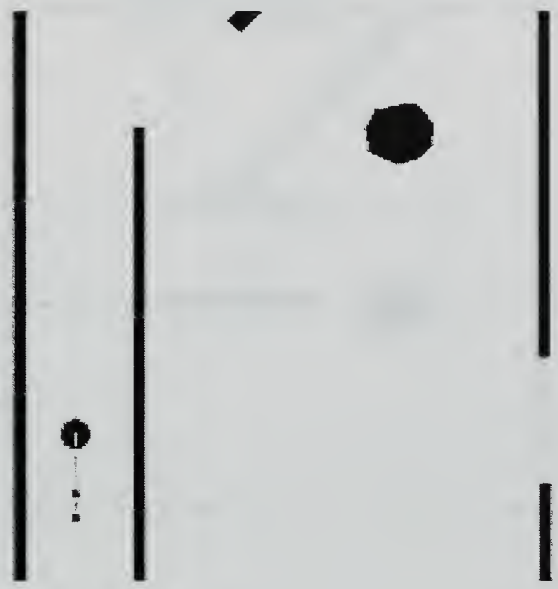

Figure 28. Simulation # 2: Problem setup

The exemplar vectors chosen by the network are shown in Table 8. In this case, the network shows a slightly greater error than in the previous example, the chosen exemplars are nearly 2 degrees from being parallel. This result seems marginally acceptable, but is also partially due to the parameters chosen for the network. Performance may be expected to differ when learning rate and tolerance parameters are adjusted.

θ	-88.4958	90.5513
ρ	26.4250	34.0822

Table 8. Simulation # 2: Exemplar vectors

The walls chosen by the network are illustrated in Figure 29, superimposed on the range data. The walls appear to coincide with the sonar data at points near the robot. The algorithm appears to work in a corridor scenario, though it is identified that accuracy might be improved if parameters are adjusted.

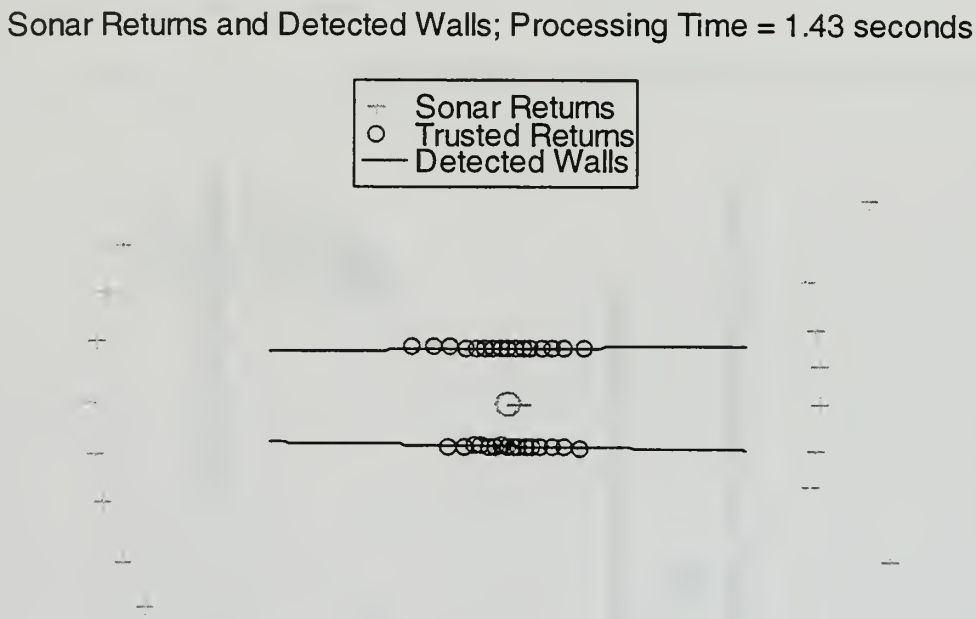

Figure 29. Simulation # 2: Sonar returns and detected walls

The computation time, as shown in Figure 29, is greater that shown for the previous scenario. This is primarily due to the fact that the two dominant clusters in the Hough domain were each comprised of a greater number of points, and loosely grouped. This implies that the neural network will take longer to cycle through a single epoch and, therefore, will likely take longer to converge. The computation time is still quite acceptable, well within the 5 second benchmark established.

3. Walls Which Are Not Orthogonal

For the next test, the robot is placed in the upper-left corner of the map, facing generally toward the corridor of the last example, as shown in Figure 30. As in the previous examples, the exemplar vectors representing the walls are shown in Table 9, and are illustrated along with the sonar returns in Figure 31.

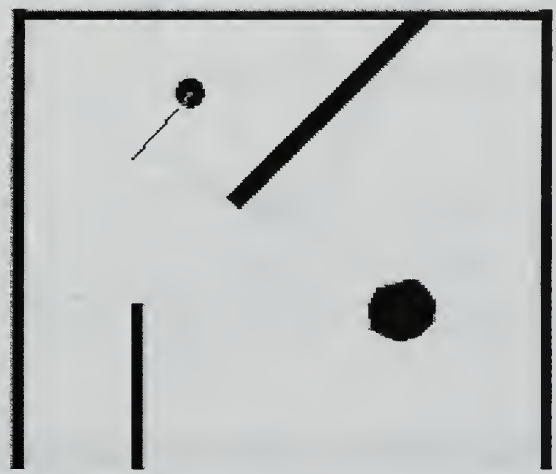

Figure 30. Simulation # 3: Problem setup

θ	72.6986	-61.4646	-153.1804
ρ	61.2021	98.7628	42.0435

Table 9. Simulation # 3: Exemplar vectors

The network has again determined the correct number of walls in its vicinity. The walls which are orthogonal in the map were chosen to be represented by exemplars which are within 2 degrees of being orthogonal. This despite the fact that one of the walls was very near the maximum trusted range of 110 inches. From Figure 31 it is apparent that the exemplar vectors chosen by the network are reasonable representations of the actual walls. The time required to process the data is also acceptable.

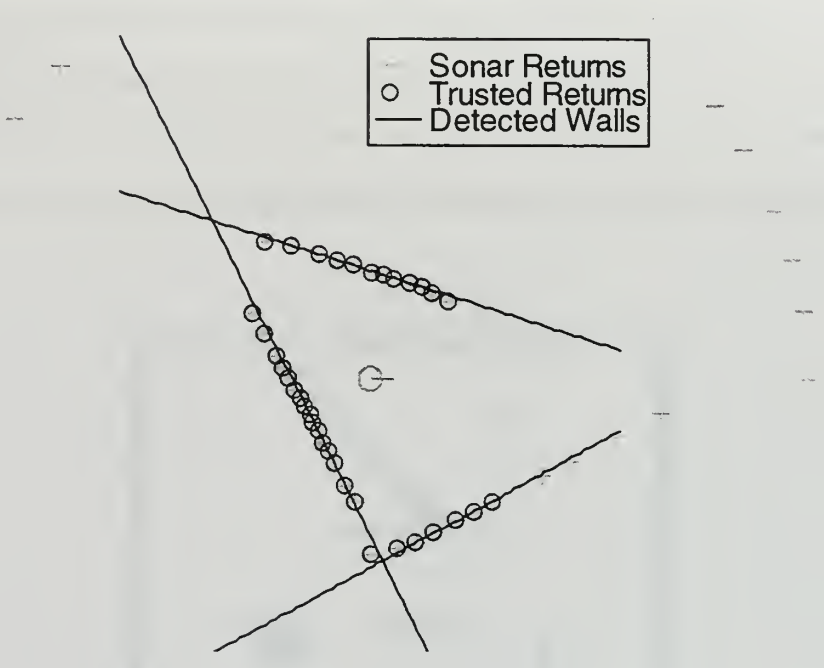

Figure 31. Simulation # 3: Sonar returns and detected walls

4. Short Walls: A Partial Failure

Another test was conducted to determine if the network could recognize and identify very short walls. For this test, the robot was placed in the upper-right corner of the map, facing generally toward the doorway, as shown in Figure 32. The wall to the left of and slightly behind the robot is short, and it is was questioned whether there would be enough echo returns from this wall for the algorithm to recognize it.

Data were collected and presented to the network in the same fashion as in the previous scenarios. The exemplar vectors chosen by the network are shown in Table 10. In this case, the network incorrectly determined that there were two walls in its vicinity. These walls are shown, plotted along with the sonar returns in Figure 33; it is clear that the short wall in question was in fact neglected by the network.

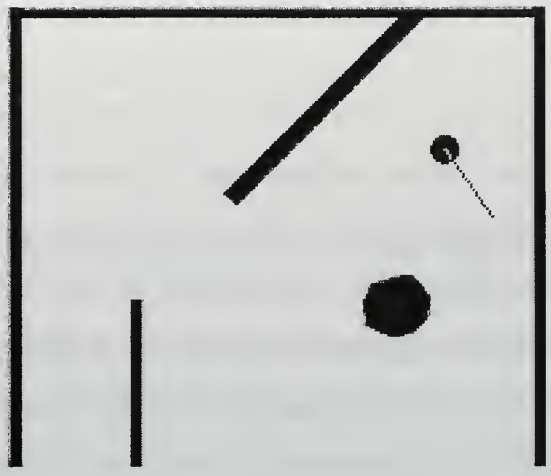

Figure 32. Simulation # 4: Problem setup

θ	-155.8561	71.8601
ρ	63.6093	53.9698

Table 10. Simulation # 4: Exemplar vectors

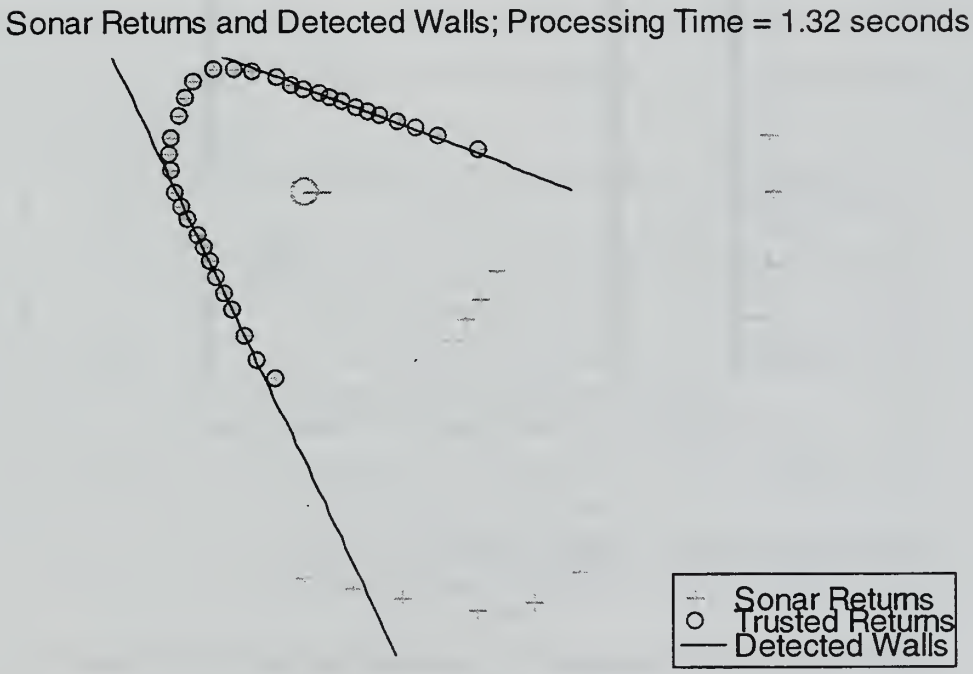

Figure 33. Simulation # 4: Sonar returns and detected walls

5. Near A Doorway: Total Failure

For any given system, it is as important to know the points of failure as success. A final simulation was conducted to determine how the network would react to a discontinuity in the wall. For this test, the robot was placed very close to an open doorway. If the robot was relatively far away from the doorway, the opening was simply ignored and a single wall was recognized. The robot was gradually moved toward the doorway until it was very close, as shown in Figure 34. In this configuration, very few echoes are returned from the wall containing the open door, and the clusters in the Hough domain become insubstantial. The neural network is unable to cluster the points, and failure occurs. In this configuration, the network did not determine any walls at all.

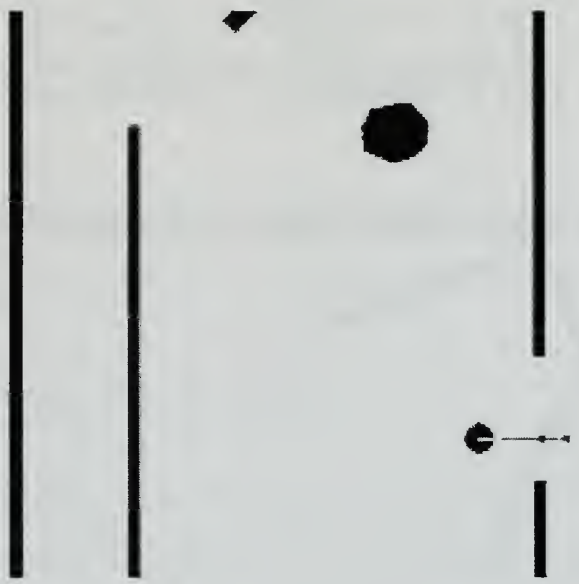

Figure 34. Simulation # 5: Problem setup

C. PROCESSING REAL WORLD SONAR DATA

Although new problems arose when the algorithm was applied to noisy sonar data collected in the real world, the overall performance remained high. Neural networks are often chosen for many applications because of their ability to perform well under noisy conditions. By checking for intersections within a neighborhood in the Hough domain,

we have also equipped that portion of the algorithm to handle a certain amount of noise. The result is an algorithm that performs nearly as well with noisy, real-world data as it does with ideal, simulated data.

1. Tuning Algorithm Parameters To Deal With Noise

When noise is present in the data, the result is inconsistency in the output of the algorithm. This problem is overcome by adjusting various parameters in the system. For example, a high learning rate in the neural network is likely to yield inconsistent results since the order of the data presented to that stage is randomized. Dropping the learning rate will improve the consistency of the algorithm, but dropping it too far will prevent the weights from converging to the clusters. Likewise, the ρ and θ tolerances used to find intersections within a neighborhood in the Hough domain must be increased if the cluster sizes are too small, and decreased if they tend to be loosely grouped. The number of neurons used in the network could also be adjusted to affect the performance of the algorithm, as well as the minimum and maximum trusted range returns. By trial and error, the values summarized in Table 11 have been found to yield consistent and accurate results with real-world data.

Symbol	Meaning	Recommended Value
p	Constant Learning Rate used to update neurons in competitive network	$0.04 < \alpha < 0.1$
θ_{TOL}	1st Tolerance used to determine if intersections in the Hough Domain are within small neighborhoods	8 degrees
ρ_{TOL}	2nd Tolerance used to determine if intersections in the Hough Domain are within small neighborhoods	6 inches
p	Number of neurons initially used in competitive network	30
NN_{TOL}	Similarity tolerance used to determine whether neurons should be combined.	0.99
R_{MAX}, R_{MIN}	Range over which sonar range returns should be considered reliable	17 to 110 inches

Table 11. Recommended parameters in algorithm

2. Real World Corner in a Cluttered Room

For the initial real-world test, the robot was placed near a corner in a somewhat cluttered room. Precise measurements from the robot center to the walls were not possible, but were also unnecessary since only consistency of the outputs is needed. One wall was in front of the robot at an orientation between 0 and 5 degrees, and at a range between 45 and 47 inches. The other wall was to the robot's left, at an orientation between 90 and 95 degrees, and at a range between 52 and 54 inches.

Three sets of 16 sonar readings were taken and presented to the network. The sonar returns were converted to (x, y) points, and those points converted to clusters in the Hough domain. The representation of the sonar returns in the Hough domain is illustrated in Figure 35. It is difficult even for a human to determine intersections from the noisy curves shown in Figure 35 (a). The task becomes somewhat easier when these curves are reduced to the clusters shown in Figure 35 (b), but even in this case the outlying data can be misleading.

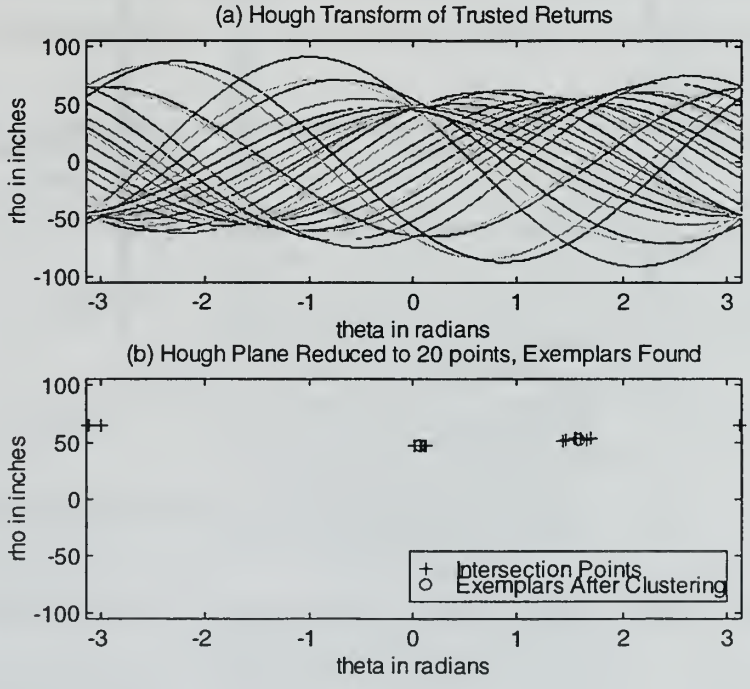

Figure 35. Hough domain representaion of sonar returns gathered in a real world corner (a) Complete curves in the Hough domain (b) Clusters in the reduced Hough domain

As shown in Figure 35 (b), the size of clusters in the Hough domain is much smaller when the sonar data are noisy. As will be seen, however, 20 points are more than enough for the competitive network to develop a consistent set of exemplar vectors.

The data points were presented to the algorithm ten times. In each case, the network (using the values given in Table 11) was able to discern exactly two walls. The resulting exemplar vectors from each presentation are shown in Table 12.

Wall 1	
θ	ρ
91.2420	52.7264
91.0209	52.0958
91.4327	52.7328
91.0269	52.8511
90.4485	52.7219
90.9137	53.0297
90.5053	53.2157
90.8307	52.9908
91.4669	52.2954
91.1774	52.6492

Wall 2	
θ	ρ
3.7852	46.6550
3.6753	46.6624
3.6506	46.6627
3.5926	46.6478
3.7589	46.6653
3.5883	46.6550
3.4605	46.6503
3.7865	46.6538
3.6721	46.6636
3.3600	46.6289

Table 12. Summary of algorithm output for sonar input gathered in real world corner

The consistency of the algorithm output is evident. The values reported for the orientation of walls range just over 1 degree over 10 samples. The values reported for the range to that wall range slightly more than an inch.

The consistency of the outputs is far more important for the chosen application than their accuracy. Localization will be addressed by having the robot find range and bearing to nearby walls at startup, and storing those values in memory. After the robot has moved about and accumulated some dead reckoning error, the robot will return to what it believes is its startup position, and take those ranges and bearings again. The dead reckoning error is taken to be the difference between the two samples. This application relies on the notion that range and bearing findings of nearby walls will be

consistent if taken from the same position. Since it is apparent that they will be consistent within approximately 1 degree and 1 inch, we may safely rely on this algorithm to correct dead reckoning errors.

The walls represented by the final set of exemplar vectors is shown in Figure 36, plotted along with the sonar returns taken from the robot's location. Although accuracy is not necessary for the chosen application, the lines appear to be fairly accurate descriptions of the sonar data collected. Finally, we note that the time required for the algorithm to develop a set of exemplar vectors is 1.21 seconds, which is well within acceptable limits.

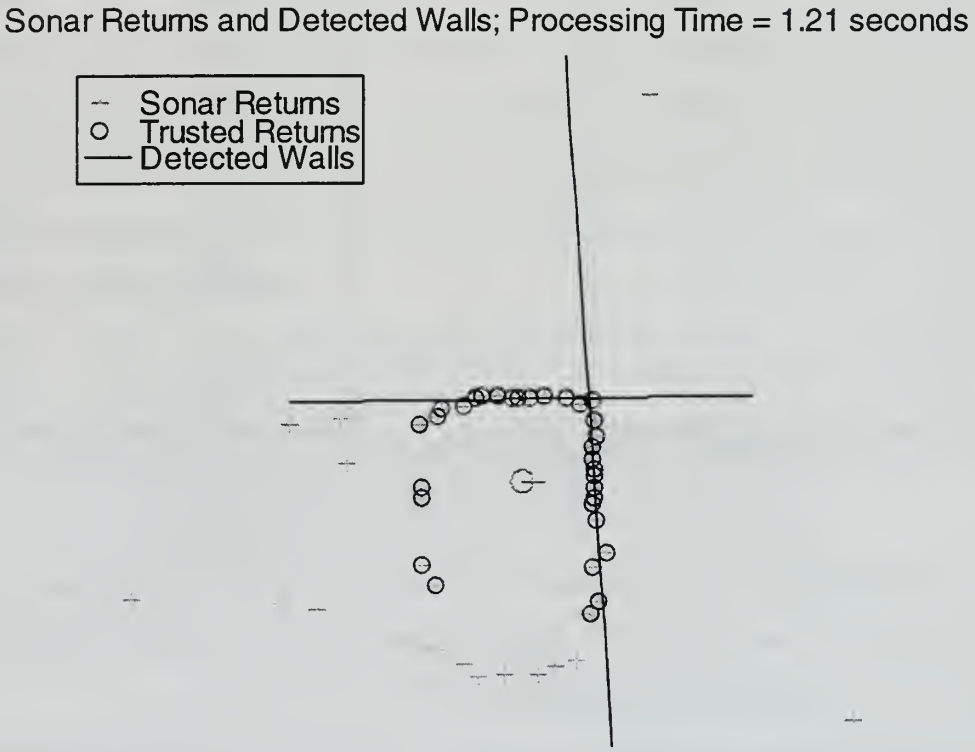

Figure 36. Sonar returns and detected walls in a real world corner

3. Real World Corridor in a Cluttered Room

For the final test, a "corridor" was constructed out of one wall in the laboratory, and scraps of cardboard taped to a countertop. No attempt was made to "smooth" the edges of the cardboard in the constructed wall. The laboratory wall also had an outcropping approximately 12 inches wide, jutting out approximately 6 inches into the room. The environment also included tables and other objects which served to obfuscate the two dimensional representation; these were intentionally left in place.

Note that for the application chosen, a corridor is not a suitable startup location. One would choose to startup the robot in a location where features are distinguishable; range and orientation to walls would ideally be identical for any location down the length of the corridor. This test is included for the sake of future research, which possibly could focus on mapping applications.

The robot gathered a set of sonar returns in the environment described. These returns are shown in Figure 37, with those returns which fell within the trusted range circled. When these data were presented to the algorithm, the two walls given by the exemplar vectors in the first row of Table 13 were found in less than a second. These results are plotted in Figure 37 along with the sonar returns. It is apparent that the walls chosen by the network are not parallel, indicating some inaccuracy in the algorithm. As stated earlier, however, consistency is more important than accuracy for this application. The data were presented to the network ten times, resulting in the exemplar vectors shown in Table 13. The consistency of the algorithm in this case is acceptable, and could possibly be improved further by dropping the learning rate and the maximum trusted range of the sonar returns. Accuracy could also be improved by dropping the maximum trusted range, and adjusting other parameters in the network as necessary. Accuracy is also greatly affected by the fact that the original range data are not entirely reliable, due to the non-ideal propagation characteristics of the acoustic signals.

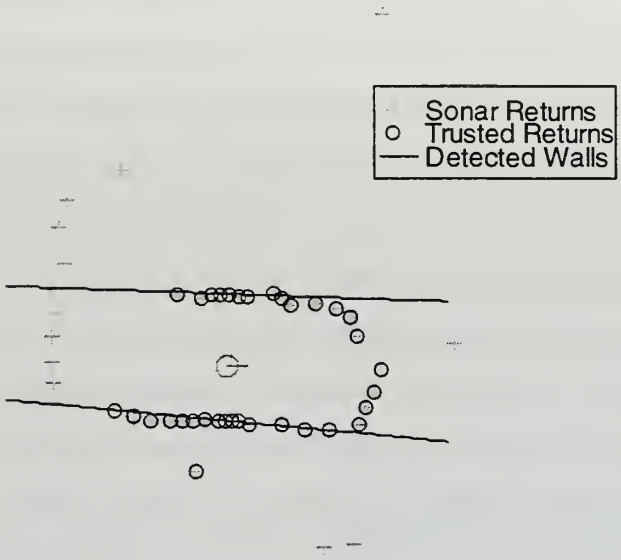

Figure 37. Sonar returns and detected walls in a real world corridor

Wall 1	
$\hat{p}$	ρ
-95.3346	36.3839
-95.8529	36.2059
-95.0802	36.7434
-95.6678	36.3207
-96.0574	35.9895
-95.8502	36.4556
-95.9314	36.3096
-95.8041	36.1182
-95.5950	36.4792
-96.1437	36.1192

Wall 2	
$\hat{p}$	ρ
88.0275	49.0765
87.8375	48.7080
87.7925	47.3405
89.0607	47.5971
88.8116	47.8388
87.2968	48.2417
87.3118	48.5150
88.0200	48.6663
89.7279	48.1172
87.0519	49.3283

Table 13. Summary of algorithm output for sonar input gathered in real world corridor

D. CHAPTER SUMMARY

In this chapter the results of testing the algorithm in both real and simulated indoor environments were presented. The algorithm was shown to perform adequately for the chosen task, although some improvement in accuracy must be achieved if the algorithm is to be applied for mapping in future research. The following chapter will discuss some of the directions this future research might take.

VIII. DISCUSSION

A. IMPLICATIONS

It is evident that, given a set of sonar echo returns from a Nomad Scout robot, the algorithm proposed in this thesis is able to determine the range to and orientation of an unspecified number of walls in the vicinity of the robot. The algorithm is able to produce results that are acceptably consistent, and can do so within an acceptable amount of time.

The immediate implication is that a robot may be commanded to determine the location of any nearby walls at startup. After some dead reckoning error has accrued, the robot may be commanded to return to the *world coordinate* origin, specified at startup. The location of nearby walls can again be determined. Any difference in the range or orientation of nearby walls can be presumed due primarily to dead reckoning error. The dead reckoning track may then be adjusted, and navigation of the robot may resume.

B. FUTURE WORK

The most pressing requirement is the implementation of the proven algorithm in C, so that it may be run in the robot's high-level control system. A more thorough analysis of the parameters specified in Table 11 should also be conducted to ensure that the parameters used are optimum.

The consistency of the algorithm could possibly be improved even further by dropping the learning rate, and adjusting the test for convergence of the neural network as necessary. In this case, it may be necessary to conduct more than 48 range findings. If the range data become redundant at more than 48 range findings, then the robot might be moved during the process. 48 samples may be taken at one location, and 48 more at another location. A thorough analysis should be conducted to determine the optimum number of samples to take, and the optimum parameters to use throughout the algorithm.

This leads directly to the concept of *continuous localization*. Since the process takes only a few seconds, there is no reason it could not be set to run in the high level

control every 30 seconds or so. Minor modifications of the code included in the appendices would be necessary to enable the algorithm to run even when the robot is far away from the *world coordinate* origin. Prior to the cycle, the dead-reckoning position of the robot would be noted. The x and y coordinates would simply be subtracted from columns 1 and 2, respectively. The steering angle from the dead reckoning track would be similarly noted, and subtracted from columns 3 and 4. In this fashion, the algorithm could be run at any arbitrary position and orientation in the *world coordinate* system.

Since the algorithm is able to place walls relative to itself, and localization provides the robot with its own location and orientation in the world, it follows that mapping applications might be explored. Mapping requires the algorithm outputs to be not only consistent, but accurate as well. This thesis has investigated only the consistency of the outputs, as the chosen application only requires this. The outputs do appear to have some accuracy, however. A thorough investigation of the accuracy should be pursued.

It is likely that the algorithm can only be as accurate as the sonar range data that are fed into it, although the Gaussian nature of the noise might dispute this claim. The accuracy of the Sensus 200 system might be improved by combining it with a time-of-flight laser [Refs. 25, 26]. Another method could entail weighting the range data according to reliability prior to or during the Hough transform [Refs. 20, 21]. Other methods could entail fusing the sensor data from the Sensus 200 with information from some other sensor system, or with range data provided by a second robot [Ref. 5].

Mapping would also require the algorithm to provide some information about the length of the wall found. It may be possible to keep track of which transducer the points in the Hough domain resulted from. This information would continue to be tracked during the clustering process. When the neural network converges and the output vectors are given, it would also be possible to specify which transducers produced data which resulted in each wall. In this fashion, some information about the length of the wall is provided; although the accuracy of this information will suffer substantially as the robot's range from the wall increases.

The suitability of this algorithm for other robotic platforms is another area that might be explored. Adaptation of the concept for platforms equipped with ranging

sensors other than ultrasonic sonar may be possible. It is also feasible that the code could be adapted to robots with alternative mobility, such as legged robots. For a robot with the proper array of sensors, it is even feasible to expand this concept to recognize features in 3 dimensions rather than 2.

Several researchers have explored the Hough transformations Cartesian shapes other than straight lines [Refs. 17, 22, 23]. It may be possible to modify the algorithm of this thesis to enable the robot to recognize features more complex than the straight walls covered in this thesis. This could eventually lead to a feature-based recognition system that works outdoors as well as indoors. Such a system would be particularly useful in underwater and space exploration scenarios, as well as cases where a land-based outdoor robot does not have access to 4 GPS satellites simultaneously or the accuracy of GPS is insufficient. If the robot were enabled to recognize complex features in 3 dimensions, the applications would be without bound.

The algorithm presented in this thesis is a demonstration of concept, not a finished product. It is intended to open to the door for follow-on projects, which will build on the fundamentals covered in this document, and bring about an enhanced degree of practicality.

```
function Points = houghred(X,Y,RTol,ThetaTol,MinRad)

% HOUGHRED  points = houghred(X,Y,RTol,ThetaTol,MinRad)
%           Returns only the key data elements of the Hough Transform
%           as a 2 x ? matrix, where each column is a key point
%           in the Hough domain. --> [Theta (Radians);
%                                     Radius (same units as X,Y)]
%
%           Hough domain is symmetric.  This function returns all
%           points in the R > 0 half of the plane, Theta is allowed
%           to range from -pi to pi.
%
%           X and Y must be row vectors of equal length representing
%           cartesian points.  R and theta will be those points where
%           Hough curves intersect near other intersections.
%
%           Possibly colinear points should be located close to each
%           other in X and Y indices (Last is adj to the first) in
%           the indexing of X and Y.  Helpful to sample ctr clockwise
%           or clockwise.
%
%           RTol, ThetaTol optional parameters; define how close an
%           intersection must be to other intersections in order to
%           be included.  Default RTol is 5, Default ThetaTol is
%           5*pi/180 radians (5 degrees).
%
%           MinRad is an optional parameter; intersections in Hough
%           domain with R < MinRad will not be included.  Default
%           value is zero.

if exist('MinRad') == 0      % assign default 'MinRad' value
   MinRad = 0;               % same units as X and Y
end

if exist('RTol') == 0        % assign default 'RTol' value
   tol = 5;                  % same units as X and Y
end

if exist('ThetaTol') == 0    % assign default 'ThetaTol' value
   tol = 5*pi/180;           % radians
end

N = length(X);               % X and Y must be equal lengths

index = [N-1,N,1:N,1,2];     % used later to make first and
                             % last indices neighbors.

PointsCount = 0;             % Initial values
Points = [0;0];

for c = 1:N

   % For each X,Y data point, find the Theta,R point where it
   % intersects its left two and right two neighbors.  If the
   % two intersections on the left are close to one another
   % (within tolerance) then include them both.  If the two
   % intersections on the right are close to one another (within
   % tolerance) then include them both.  For 1st data point,
   % left neighbors are the last two points.  For last data
   % point, the first two indices are its right neighbors.

   % Find the theta values of the intersections
```

```
   L2Th = atan2(X(index(c))-X(c),Y(c)-Y(index(c)));
   L1Th = atan2(X(index(c+1))-X(c),Y(c)-Y(index(c+1)));
   R1Th = atan2(X(index(c+3))-X(c),Y(c)-Y(index(c+3)));
   R2Th = atan2(X(index(c+4))-X(c),Y(c)-Y(index(c+4)));

   % If the points are colinear, then L1Th should approximately
   % equal L2Th and R2Th should approximately equal R1Th.  Also,
   % Left theta's should be approximately pi radians away from
   % Right theta's.  Check to see if this is true.

   Theta_check = 0;
   R_check =     0;
   if sqrt((L2Th - L1Th)^2) < ThetaTol
      if sqrt((R2Th - R1Th)^2) < ThetaTol
         if sqrt((sqrt((L2Th - R2Th)^2) - pi)^2) < ThetaTol
            Theta_check = 1;
         end
      end
   end

   % Only compute R values if thetas were in the same neighborhood.

   if Theta_check == 1

      L2R  = X(c)*cos(L2Th) + Y(c)*sin(L2Th);
      L1R  = X(c)*cos(L1Th) + Y(c)*sin(L1Th);
      R2R  = X(c)*cos(R2Th) + Y(c)*sin(R2Th);
      R1R  = X(c)*cos(R1Th) + Y(c)*sin(R1Th);

      % Should have approximately equal R values on the left and
      % approximately equal on the right.  Left R values should be
      % approximately -1 * Right R values.

      if sqrt((L2R - L1R)^2) < RTol
         if sqrt((R2R - R1R)^2) < RTol
            if sqrt(((-1*L2R) - R2R)^2) < RTol
               R_check = 1;
            end
         end
      end
   end

   if Theta_check == 1
      if R_check == 1
         PointsCount = PointsCount + 4;
         Points(1,PointsCount-3) = L2Th;
         Points(2,PointsCount-3) = L2R;
         Points(1,PointsCount-2) = L1Th;
         Points(2,PointsCount-2) = L1R;
         Points(1,PointsCount-1) = R1Th;
         Points(2,PointsCount-1) = R1R;
         Points(1,PointsCount)   = R2Th;
         Points(2,PointsCount)   = R2R;
      end
   end
end

% Hough domain is symmetric.  We deal only w/ -pi < theta < pi
% and R > 0.  Points with R < 0 must be shifted.

% all points w/ R<0 must be shifted to upper half of plane
% and shifted by pi radians

for c = 1:size(Points,2)                % for each point
```

```
   if Points(2,c) < 0                    % if R < 0
      if Points(1,c) < 0                    % if Theta < 0
         Points(1,c) = Points(1,c) + pi;       % Add pi to theta
         Points(2,c) = Points(2,c)*(-1);       % Mult R by -1
      else                                  % Theta >= 0
         Points(1,c) = Points(1,c) - pi;       % Subtract pi from theta
         Points(2,c) = Points(2,c)*(-1);       % Mult R by -1
      end
   end
end

% Lastly, ignore any points with R < MinRad

for c = 1:size(Points,2)
   if Points(2,c) < MinRad
      Points(:,c) = [NaN;NaN];           % set points w/ R < MinRad
   end                                   % equal to NaN
end

includes = find(Points(1,:));            % Throw out all the NaN's
Points = Points(:,includes);
```

```
function exemplars = nnclust(X_in,Y_in,range,p,alpha,tol)

%NNCLUST  Function to find an unspecified # of clusters in
%         2 Dimensions.  For thesis.
%
%         E = nnclust(X,Y,range,p) returns an 2 x n matrix, where
%         n is the # of clusters found, and each row is an exemplar
%         vector representing the approximate center of the cluster.
%
%         X & Y are row vectors of equal length, and each X,Y
%         pair is a data point to be analyzed.  Required.
%
%         range = [Xmin,Xmax,Ymin,Ymax] is the range over which data
%         should be expected to appear.  Default is max & min values.
%
%         p is the number of neurons to use, should be approximately
%         10 times the number of clusters expected. Default = 30.
%
%         alpha is learning rate, a vector the same length as X and Y.
%         default is 0.5
%
%         tol is an optional parameter between 0 and 1 specifying how
%         similar vectors should be before they are combined into a
%         single vector.  1 is identical, 0 is orthogonal.  Default
%         value is 0.999 (Dot-product similarity)

rand('seed',sum(100*clock));   % Sets new value for rand seed each time

% DEFINITION OF DEFAULT VALUES FOR PARAMETERS

if exist('range') == 0
   range = [min(X_in),max(X_in),min(Y_in),max(Y_in)];
end

if exist('p') == 0
   p = 30;
end

if exist('alpha') == 0
   alpha = 0.5*ones(size(X_in));
end

if exist('tol') == 0
   tol = 0.999;
end

% NORMALIZATION PROCESS -- First, restrict analysis to the unit square

X_norm = max([-1*range(1),range(2)]);
Y_norm = max([-1*range(3),range(4)]);

X = X_in/X_norm;  % -1<X<1
Y = Y_in/Y_norm;  % -1<Y<1

Nsq = 2;   % Max length of any vector is sqrt(2).

% Now add a third dimension (unit sphere) so each input has length 1

Input = [X;
         Y;
         sqrt(Nsq - (X.^2 + Y.^2))]'/sqrt(Nsq);
```

```
% WEIGHT INITIALIZATION

Xr = ((range(2)-range(1))*rand(1,p)+range(1))/X_norm;
Yr = ((range(4)-range(3))*rand(1,p)+range(3))/Y_norm;

% Weights must also be on unit sphere, over same range as data.

W = [Xr;
     Yr;
     sqrt(Nsq - (Xr.^2 + Yr.^2))]/sqrt(Nsq);

% W is now a 3 x p matrix, each column is a random vector uniformly
% distributed over the same portion of the unit sphere as the
% data to be clustered.

consecutive   = 0;                % initial values
convergence   = 0;
presentations = 0;

while convergence == 0

   Ticker = zeros(1,size(W,2));   % initial value

   for i = 1:length(X)
      S = Input(i,:)*W;
      [useless,c] = max(S);

                            % c is the neuron that won.
                            % If there was a tie, the vector with the
                            % lowest index won.

      Ticker(c) = Ticker(c) + 1;

      % UPDATE WEIGHTS (Must retain normalization, so need to adjust
      % the third dimension

      W(:,c) = W(:,c) + alpha(i)*(Input(i,:)' - W(:,c));
      W(:,c) = W(:,c)/sqrt(W(1,c)^2 + W(2,c)^2 + W(3,c)^2);
   end  % end for loop

   % DELETE UNUSED NEURONS (WEIGHTS)

   [useless,keepers] = find(Ticker>0);
   possibles = W(:,keepers);
   Ticker = Ticker(keepers);
   delete_counter = size(W,2)-size(possibles,2);

   % COMBINE SIMILAR VECTORS INTO ONE AND CREATE A NEW RANDOM WEIGHT
   combine_counter = 0;
   for i = 1:size(possibles,2)-1
      [Y,sim] = max(diag(possibles'*possibles,i));
      if Y > tol
         combine_counter = combine_counter + 1;
         temp1 = Ticker(sim)*possibles(:,sim);
         temp2 = Ticker(sim+i)*possibles(:,sim+i);
         temp3 = Ticker(sim)+Ticker(sim+i);
         possibles(:,sim) = (temp1 + temp2)/temp3;
         xnew = ((range(2)-range(1))*rand(1,1)+range(1))/X_norm;
         ynew = ((range(4)-range(3))*rand(1,1)+range(3))/Y_norm;
         possibles(:,sim+i) = [xnew;
                               ynew;
                               sqrt(Nsq-(xnew^2 + ynew^2))]/sqrt(Nsq);

      end  % end if statement
   end  % end for loop
```

```
   % Check to see if we've converged yet...

   presentations = presentations + sum(Ticker);
   if presentations > 200                              % Minimum 200
presentations
      if (delete_counter + combine_counter == 0);      % If no weights were
                                                       % deleted or
                                                       % combined above
         consecutive = consecutive + 1;                % for two epochs
         if consecutive > 1                            % in a row assume
            convergence = 1;                           % convergence.
         end  % end if statement
      else
         consecutive = 0;
      end  % end if/else
   end  % end if statement

   W = possibles;          % Go back thru with any surviving weights
end  % end while statement

% Discard small clusters (less than # of data points / # of weights).

[useless,keepers] = find(Ticker>(length(X_in)/p));

% Put output back in the 2-D form that the input was in.

exemplars = [W(1,keepers)*X_norm;
             W(2,keepers)*Y_norm]*sqrt(Nsq);
```

APPENDIX C. FINDWALL.M

```
function walls = findwall(FileName)

%FINDWALL  walls = findwall('filename.dat')
%
%          Specific application for thesis.  'filename.dat' is the
%          name of a file containing range findings from a NOMAD
%          SCOUT robot.  Format should be a 'dat' file (ASCII).
%          data must be arranged in 5 columns as shown below, and
%          number of rows should be an integer multiple of 16.
%          Function assumes sonar hardware configuration is NOMAD
%          default: i.e. 16 sonars equally spaced about circum-
%          ference, range findings taken counter clockwise.
%
%          Finds range and bearing to the closest point
%          of any significant walls near the robot, in robot
%          coordinates.
%
%          Output is a 2 x ? matrix; each column represents a
%          Theta,R pair indicating the presence of a wall.  The top
%          row is Theta in degrees, the bottom row is R in inches.
%
%          For code simplicity sake, this version requires the file
%          name extension to be EXACTLY ".dat".
% ------------------------------------------------------------------
%          DATA FORMAT:  Col 1 = X-position of robot times 10
%                        Col 2 = Y-position of robot times 10
%                        Col 3 = Steering Angle in degrees times 10
%                        Col 4 = Turret Angle in degrees times 10
%                        Col 5 = range return of ith sonar in inches
% ------------------------------------------------------------------

eval(['load ',FileName]);   % Load Data
input = eval(FileName(1:size(FileName,2)-4));% Name it 'input'

% Useful Constants-----------------------------------------------------

RobotRadius = 7.185;         % inches -- Nomad Scout
shift = 0:22.5:(360-22.5);   % angle of ea sonar relative to T-angle
HTolR  = 6;                  % Tol for Hough reduction - radius (in)
HTolTh = 8*pi/180;           % Tol for Hough reduction - Theta (rad)
p = 30;                      % # of neurons to start with in NN
NNTol = 0.99;                % Similarity tolerance for NN
ConstLR = 0.05;              % Learning Rate for NN (if constant)
max_trusted = 110;           % Largest sonar return to be trusted
min_trusted = 17;            % Smallest sonar return to be trusted

% SORT & CONDITION THE INPUTS------------------------------------------

% We don't need steering angle AND turret angle if we're working with
% a scout; Col # 4 is meaningless.  So we'll turn column 4 into the
% ACTUAL angle of each range finding relative to turret angle.
% 16 Sonars are equally spaced 22.5 degrees apart, or as defined in
% the variable 'shift' above.  NOTE: if the number of rows in the
% input file is not an integer multiple of the number of sonars
% defined in 'shift' (default 16), an error message will result.

for c = 1:size(input,1)/length(shift)
   input((16*(c-1)+1):16*c,4)=input((16*(c-1)+1):16*c,3)/10+shift';
end

% Now, go back through the column 3 we created, and make sure all
% angles are between 0 and 360.
```

```
for c = 1:size(input,1)
   if input(c,4)<0
      input(c,4) = input(c,4)+360;
   elseif input(c,4)>= 360
      input(c,4) = input(c,4)-360;
   end
end

% Finally, sort the entire set of range findings counter-clockwise

[useless_vector,I] = sort(input(:,4));
input = input(I,:);
clear useless_vector

% Ignore range findings too small or too big

in = input;

for row = 1:size(in,1)
   if in(row,5) > max_trusted
      in(row,5) = NaN;
   elseif in(row,5) < min_trusted
      in(row,5) = NaN;
   end
end

I = find(in(:,5));
in = in(I,:);

% GET X,Y LOCATIONS OF ALL SONAR RETURNS TRUSTED--------------------

XY = zeros(2,size(in,1));
for c = 1:size(in,1)
   Xrobot = in(c,1)/10;
   Yrobot = in(c,2)/10;
   range = RobotRadius + in(c,5);
   angle = in(c,4)*pi/180;
   X = Xrobot + range*cos(angle);
   Y = Yrobot + range*sin(angle);
   XY(:,c) = [X;Y];
end

% TAKE REDUCED HOUGH TRANSFORM OF X,Y PAIRS------------------------

cl_rad = houghred(XY(1,:),XY(2,:),HTolR,HTolTh,RobotRadius+min_trusted);
cl_deg = [cl_rad(1,:)*180/pi;
          cl_rad(2,:)];

[Y,I] = sort(rand(1,size(cl_deg,2)));       % Put the points in rand
cl = cl_deg(:,I);                           % order for presentation

% FIND THE CLUSTERS IN HOUGH DOMAIN--------------------------------

range = [-180,180,RobotRadius+min_trusted,max_trusted];
alpha1 = ConstLR*ones(size(cl,2));
walls = nnclust(cl(1,:),cl(2,:),range,p,alpha1,NNTol);
```

```
/*************************************************************
 *
 * PROGRAM: gather.c
 *
 * PURPOSE: To collect sonar data for later off-line processing
 *          to locate walls.  Modified for Scout
 *************************************************************/


/*** Include Files ***/

#include "Nclient.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


/***  Conversion MACROS courtesy of Nomadic Inc   ***/

#define RIGHT(trans, steer)  (trans + (int)((float)steer*368.61/3600.0))
#define LEFT(trans, steer)   (trans - (int)((float)steer*368.61/3600.0))

#define scout_vm(trans, steer)   vm(RIGHT(trans, steer), LEFT(trans,
steer), 0)
#define scout_pr(trans, steer)   pr(RIGHT(trans, steer), LEFT(trans,
steer), 0)


/*** Function Prototypes ***/

void GetSensorData(void);


/*** Global Variables ***/

long SonarRange[16];       /* Array of sonar readings (inches) */
long IRRange[16];          /* Array of infrared readings (no units) */
long robot_config[4];      /* Array - robot configuration */

/*** Main Program ***/

main (unsigned int argc, char** argv)
{
   int i, j, index;
   int order[16];
   FILE *fp;

   /* Connect to Nserver. The parameter passed must always be 1. */
   SERV_TCP_PORT = 7020;
   connect_robot(1, MODEL_SCOUT, "scout1.ece.nps.navy.mil", 4001);


   /* Initialize Smask and send to robot. Smask is a large array that
      controls which data the robot returns back to the server. This
      function tells the robot to give us everything. */
  init_mask();


   /* Configure timeout (given in seconds). This is how long the robot
      will keep moving if you become disconnected. Set this low if there
      are walls nearby. */
   conf_tm(1);
```

```
   /* Sonar setup.  As you look at robot from top, Sonar 0 is the one
      in the direction the robot is facing.  Then they number counter
      clockwise up to 15.  */
   for (i = 0; i < 16; i++)
      order[i] = i;
   conf_sn(15,order);



   fp = fopen("range.dat", "w");

   /* Guts of the program. To make robot rotate 7.5 degrees, the
      command is scout_vm(0,75).  The direction will be counter
      clockwise as you view the robot from the top.  Need to
      GetSensorData, rotate, GetSensorData again, rotate again,
      GetSensorData a third time, then return to original state.*/

   GetSensorData();

   for (j=0; j<16; j++)
      fprintf(fp, "%8d %8d %8d %8d %8d  \n",
         robot_config[0],robot_config[1],robot_config[2],
         robot_config[3],SonarRange[j]);

   scout_vm(0,75);

   sleep(5);

   GetSensorData();

   for (j=0; j<16; j++)
      fprintf(fp, "%8d %8d %8d %8d %8d  \n",
         robot_config[0],robot_config[1],robot_config[2],
         robot_config[3],SonarRange[j]);


   scout_vm(0,75);

   sleep(5);

   GetSensorData();

   for (j=0; j<16; j++)
      fprintf(fp, "%8d %8d %8d %8d %8d  \n",
         robot_config[0],robot_config[1],robot_config[2],
         robot_config[3],SonarRange[j]);

   scout_vm(0,-150);

   fclose(fp);

   /* Disconnect. */
   disconnect_robot(1);
}



/* GetSensorData(). Read in sensor data and load into arrays. */
void GetSensorData (void)
{
   int i;

   /* Read all sensors and load data into State array. */
```

```
   gs();

   /* Read State array data and put readings into individual arrays. */
   for (i = 0; i < 16; i++)
      {
      /* Sonar ranges are given in inches, and can be between 6 and
         255, inclusive. */
      SonarRange[i] = State[17+i];

      /* IR readings are between 0 and 15, inclusive. This value is
         inversely proportional to the light reflected by the detected
         object, and is thus proportional to the distance of the
         object. Due to the many environmental variables effecting the
         reflectance of infrared light, distances cannot be accurately
         ascribed to the IR readings. */
      IRRange[i] = State[1+i];
      }

   for (i = 0; i < 4; i++)
      robot_config[i] = State[34+i];
}
```

REFERENCES

1. "Chernobyl DOE, NASA, and Industry Team Up to Create Damaged Reactor Model," *Nuclear Waste News*, April 1998.

2. Baker, S. and Matlack, C., "Chernobyl: If You Can Make It Here...," *Business Week*, March 30, 1998.

3. Lange, L., "Mars mission spotlights robotics' potential, but Japan owns the market – U.S. plays catch-up as robots prove their mettle," *Electronic Engineering Times*, February 23, 1998 (correction appended March 16, 1998).

4. Hernandez,G., *An Integrated GPS/INS Navigation System for Small AUVs Using An Asynchronous Kalman Filter*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1998.

5. Hillmeyer, P., *Implementation of a Multiple Robot Frontier-Based Explorations System as a Testbed for Battlefield Reconnaissance Support*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1998.

6. Albayrak, O., *Line and Circle Formation of Distributed Autonomous Mobile Robots with Limited Sensor Range*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1996.

7. Mays, E. and Reid, F., *Shepherd Rotary Vehicle: Multivariate Motion Control and Planning*, Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1997.

8. "Robotic Digest – Defense Advanced Research Projects Agency," *Military Robotics* August 22, 1997.

9. H. R. Everett, *Sensors for Mobile Robots: Theory and Application*, A. K. Peters, Ltd., Wellesley, MA, 1995, pp. 1-65.

10. *Scout Beta 1.1*, Nomadic Technologies, Inc., Mountain View, CA, 1998.

11. *The Nomad Scout*, Nomadic Technologies, Inc., Mountain View, CA, July 1997.

12. Nomadic Technologies, Inc., Webpage, http://www.robots.com

13. *Nomad 200 Hardware Manual*, Nomadic Technologies, Inc., Mountain View, CA, February 1997.

14. Hough, P. V. C., "A Method and Means for Recognizing Complex Patterns," U. S. Patent No. 3,069,654, 1962.

15. Hagan, M., Demuth, H.,and Beale, M. *Neural Network Design*, PWS Publishing, Boston, MA, 1996.

16. Lin, C. and Lee, C. *Neural Fuzzy Systems, A Neuro-Fuzzy Synergism to Intelligent Systems*, Prentice-Hall, Upper Saddle River, 1996.

17. Duda, R. O. and Hart, P. E. "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, 15(1):11-15, 1972.

18. Choy, C., Ser, P., and Siu, W., "Peak detection in Hough transform via self-organizing learning," *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 139-142, Seattle, WA, May 1995.

19. Forsberg, J., Larsson, U., Ahman, P., and Wernersson, A., "The Hough transform inside the feedback loop of a mobile robot," *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 791-798, Atlanta, GA, May 1993.

20. Forsberg, J., Larsson, U., and Wernersson, A., "Mobile robot navigation using the range-weighted Hough transform," *IEEE Robotics and Automation Magazine*, 2(1):18-26, March 1995.

21. Larsson, U., Forsberg, J., and Wernersson, A., "Mobile robot localization: Integrating measurements from a time-of-flight laser," *IEEE Transactions on Industrial Electronics*, 43(3):422-431, June 1996.

22. Yuen, K., and Chan, W., "A solution to the generalized Duda and Hart Problem using Fourier parameterization," *Proceedings of the IEEE International Conference on Speech, Image Processing, and Neural Networks*, pages 441-444, Hong Kong, April 1994.

23. Chan C. K., and Sandler, M. B., "A complete shape recognition system using the Hough transform and neural network." *Proceedings of the Eleventh IEEE International Conference on Pattern Recognition*, (Conference B: Pattern Recognition Methodology and Systems) pages 21-24, The Hague, Netherlands, September 1992.

24. Latt, K., *Sonar-Based Localization of Mobile Robots Using the Hough Transform*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1997.

25. Yamauchi, B., Schultz, A., and Adams, W., *Integrating Exploration and Localization for Mobile Robots*, Report AIC 97-021 Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington D. C., 1997.

26. Yamauchi, B., Schultz, A., and Adams, W., "Mobile Robot Exploration and Map-Building with Continuous Localization," *Proceedings of the IEEE International Conference on Robotics and Automation*, Lueven, Belguim, May, 1998.

27. Bardash, M. J., "Three dimensional imaging system using laser generated ultrashort x-ray pulser," U. S. Patent number 5,703,923, 1997.

28. Wildes, R., Asmuth, J., Hanna, K., Hsu, S., Kolczynski, R., Matey, J., McBride, S., "Automated, non-invasive iris recognition system and method," U.S. Patent number 5,572,596.

29. Duda, R. O., and Hart, P. E., *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, NY, 1973.

30. Kohonen, T., *Self-Organization and Associative Memory*, 3rd ed. Springer-Verlag, New York, NY, 1989.

31. Dayhoff, J., *Neural Network Architectures, An Introduction*, Van Nostrand Reinhold Publishing, New York, NY, 1990.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   300 Russell Road
   Quantico, VA 22134-5130

6. Marine Corps Representative 1
   Naval Postgraduate School
   Code 037 Bldg. 234 HA-220
   699 Dyer Road
   Monterey, CA 93943

7. Marine Corps Tactical Systems Support Activity 1
   Technical Advisory Branch
   Attn: Maj J. C. Cummiskey
   Camp Pendleton, CA 92055-5080

8. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

9. Professor Xiaoping Yun, Code EC/YX 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121

10. Professor Robert G. Hutchins, Code EC/HU 1
    Department of Electrical and Computer Engineering
    Naval Postgraduate School
    Monterey, CA 93943-5121

11. Captain Jonathan S. Glennon, USMC 3
    2010 Lakeway Drive
    Holland, MI 49423